

Demonstrating RINA using the GENI Testbed

Yuefeng Wang

Flavio Esposito

Ibrahim Matta

Computer Science Department, Boston University

Boston, MA 02215

{wyf, flavio, matta}@cs.bu.edu

Abstract—The inability of the current Internet architecture to accommodate modern requirements has spurred novel designs for future Internet architectures. The Global Environment for Network Innovations (GENI) is a wide-area virtual network testbed which allows experimentation of such architectures for possible deployment. We have contributed to the efforts of redesigning the Internet with a Recursive InterNetwork Architecture (RINA), and in this paper we demonstrate its practicability by running a prototype on the GENI testbed.

We focus on testing two fundamental features of our architecture: security and manageability, discussing in detail how the experimentation was carried, and pointing out some lessons learned using the testbed.

I. INTRODUCTION

The lack of a complete Internet architecture continues to dramatically increase the management costs and complexity of all distributed systems. Fundamentally, the problem with the current Internet architecture is that a layer expects a certain level of service from the underlying layer, but the underlying layer may not meet that requirement. This inability of layers to communicate what service level is needed, together with the desire to offer virtualized network services, have compounded existing network service management challenges. In response to such challenges, the research community has directed significant efforts and resources toward *clean slate* architecture design.

Even when a novel architecture solution is well-tested and accepted by the research community, the barrier to deployment remains a tremendous challenge. The Global Environment for Network Innovations (GENI) initiative [6] has been designed to overcome such challenges, supporting research experiments in computer networking and related disciplines. GENI experimenters are in fact able to acquire isolated (layer-2) virtual “slices” of a physical network spanning resources across multiple federated aggregates, *e.g.*, those provided by existing virtual network testbeds [8], [9], [10].

We participate in the community efforts of re-architecting the Internet, by demonstrating how our *Recursive InterNetwork Architecture (RINA)* [15] prototype can run using GENI resources.

RINA is based on the fundamental principle that networking is *Inter-Process Communication (IPC)*. RINA recurses the IPC service over different scopes. Specifically, a scope defines a *Distributed IPC Facility (DIF)* comprised of the set of IPC processes, possibly running on different machines, that collaboratively provide a set of well-defined flow services to upper application processes. The mechanisms within each IPC process are the same but each DIF layer is instantiated using

a possibly different set of policies. For example, two low-level DIFs, DIF1 and DIF2, could provide communication services to a higher level DIF3, where the IPC processes of DIF1 run their own management policies (*e.g.*, authentication, addressing, routing, resource allocation) and the IPC processes of DIF2 run the same mechanisms, instantiated with different policies. And DIF3, building on DIF1 and DIF2 services, runs its own different set of policies. A DIF is thus a policy-intantiated service block that can be stacked to provide wider scoped services. A more detailed RINA background is given in Section II.

We focus our GENI demonstration on two main aspects of the architecture: (i) enrolling/authenticating an IPC process into an existing DIF, to form a secure layer, and (ii) the instantiation of a new, higher level DIF layer on top of two preexisting DIFs (possibly instantiated with different policies.) The details of the experiments are in Section V.

In designing and running our GENI experiments, we faced some difficulties when trying to reserve slices across different aggregates, as a layer-2 tunnel could not always be reserved. Often reserving a slice with complex topologies required several attempts. Detailed comments on our GENI experience are in Section VI.

II. RECURSIVE INTERNETWORK ARCHITECTURE

Recursive InterNetwork Architecture (RINA) [1] is a new Internet architecture based on the fundamental design principle that *networking is Inter-Process Communication (IPC)*. A collection of distributed IPC processes with shared states is called a *Distributed Inter-process communication Facility (DIF)*. A DIF *layer* provides data transport services over a certain scope (*i.e.*, range of operation).

Each IPC process consists of three sets of mechanisms dealing with communication aspects at different time-scales: (i) data transfer, *e.g.* fragmentation, (ii) data transfer control, *e.g.*, flow control, and (iii) management, *e.g.*, routing. The mechanisms also operate over different granularities and time scales: data transfer (error and flow) control mechanisms for example are flow-specific and they operate at a round trip time scale, while the IPC management deals with wide-area shared states (*e.g.* routing table updates) and operates over longer time scales. DIF layers can be discovered and created on the fly [11], and their mechanisms are able to respond and support policy adaptation as network states (*e.g.* quality of underlying services) change.

RINA Prototype: the design and implementation of RINA did not arise *de novo*, but instead derives from first principles [1], [2], and combines design and implementation efforts across different institutions [13]. This is in contrast with most clean-slate Internet architecture prototypes that rely on a single implementation.

We have been adding new components to our prototype for over two years. The current version [15] resulted in about 30,000 lines of Java code, excluding support libraries, test applications, and configuration files. Before testing on GENI, it was tested with IPC processes running on our local campus network, both on Linux machines and on Android phones.

All management and data transfer messages are serialized and deserialized using the Google Protocol Buffer [14]. The prototype is built as an overlay over the Java TCP library, where TCP connections are used to emulate physical connectivity. We do not rely on TCP/IP protocols to transport and route packets among communicating processes of a DIF.

III. EXPERIMENT SETTINGS

In our experiments, we used instances of physical resources provided by ProtoGENI [7], a prototype implementation and deployment of the GENI testbed. ProtoGENI is the Control Framework for GENI Cluster C, largely based on the Emulab software and Slice-Based Facility Architecture [16]. To request their virtual resources, *e.g.*, virtual nodes, virtual links, interfaces and other metadata (such as operating system types or end-to-end delays), ProtoGENI users need to use the `RSpec`, an XML-based specification language following the RELAX NG Compact syntax [16].

In GENI, a *slice* is the resource container in which users can run an experiment, and a *sliver* is the virtual instance of the physical resource (*e.g.* a virtual machine or a virtual LAN) granted to the user. To reserve virtual resources in GENI, we hence need to reserve a slice, and then add at least one sliver.

A. Flack and Omni

To set up an experiment, we used both Flack [17] and Omni [18]. Flack is a Flash-based web interface for viewing and allocating ProtoGENI, as well as other supported GENI resources. Omni is a command line tool for finding and reserving all GENI resources.¹

Flack provides a more direct and visible way to build the experiment environment as it is a GUI tool. However, when we used Flack to reserve resources, the process is slower and often returned errors across several browsers. Not surprisingly, Omni is less user-friendly but faster both in terms of resource reservation response and, thanks to its API, in reproducing experiments with scripting. The only drawback of Omni is the need to create the `RSpec` file manually to build the experiment topology. We used Omni to reserve the resources for our experiments.

¹ProtoGENI resources are a subset of all GENI resources: only those present in the single cluster C.

IV. EXPERIMENT SETUP USING OMNI

In order to be able to use the GENI testbed, we need to create an account, and acquire credentials with public/private keys and SSL certificate. After installing and configuring the Omni python scripts, we reserved the GENI resources as follows:

Commands used:

- a. \$ `omni.py createslice rina`
- b. \$ `omni.py listresources -a pg-utah -o`
- c. \$ `omni.py createsliver -a pg-utah rina rina.rspec`
- d. \$ `omni.py listresources -a pg-utah rina`
- e. \$ `omni.py sliverstatus -a pg-utah rina`

We first created a slice using the `createslice` (command a) with `rina` as the slice name. Then, we list the available resources from `Utah` Aggregate Manager (AM) using `listresources` (command b). After getting the details of all the available resources on the AM, we created the `RSpec` file with file name `rina.rspec`, which describes the resources we will use (including the nodes and the links). Next, we reserve the resources using `createsliver` (command c). Once the resources are reserved, we can see them using `listresources` (command d), or we can check the status using `sliverstatus` (command e). The resources reserved using Omni can also be seen in Flack.

When the resources are successfully reserved, Omni returns the names of all reserved nodes (machines). Then we can use SSH to log into these nodes and upload our own code to run the experiment. Our program is in java, so we installed the Java environment as follows:

```
$ sudo yum install java-1.6.0-openjdk
```

Once the Java environment is setup, we can start our RINA experiments.

V. RINA EXPERIMENTS

We have run two kinds of RINA experiments on ProtoGENI. One is the enrollment/authentication procedure of a new IPC process with an existing DIF, and the other is the formation of a higher-level DIF on top of two pre-existing DIFs. The new DIF is created on the fly to provide communication service to two applications which have no common underlying DIF to communicate at the beginning.

We have chosen to test these two experiments, as they represent two key features of our RINA architecture. In RINA, application processes communicate by means of an IPC process within a DIF. IPC processes need to be enrolled into the same DIF to communicate. All IPC processes must be authenticated using an explicit enrollment procedure. When an application process cannot find an IPC process that can provide such communication service, a new DIF needs to be dynamically created where an IPC process in the new DIF could provide the communication service needed.

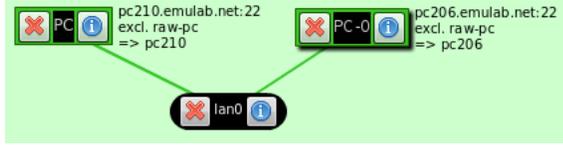


Fig. 1. Enrollment procedure slice: network topology seen in Flack.

A. Enrollment procedure

The enrollment, the allocation (*i.e.*, establishment of a flow) and the data transfer phases are the three *phases of operation* that need to be performed for a sender-receiver communication to occur. An enrollment creates, maintains, distributes (and eventually deletes upon withdrawal) information within a DIF. Such information may include addressing, access-control rules or other management policies necessary to create instances and characterize a communication.

The enrollment procedure starts with an enrollment request to the DIF manager who is responsible for enrolling new members into the DIF. The DIF manager requests authentication (*e.g.* with user and password or SSH keys) from the requesting IPC process. If successfully authenticated, the DIF manager assigns the new member a name – a private address within the DIF, and other DIF configuration states, such as routing policy, the current members of the DIF, and the list of application processes directly reachable by them. Lastly, the DIF manager updates all the other existing IPC processes with the new member information, such as new application processes reachable from it. After the enrollment procedure is complete, the new member is able to communicate with all members in the same DIF.

Figure 1 is the network topology displayed in Flack after we reserve two nodes using Omni from the Utah aggregate. Node PC (*pc210.emulab.net*) is the DIF Manager, and Node PC-0 (*pc206.emulab.net*) is the new IPC process which is going to join the DIF. These two nodes are connected by LAN lan0. We first run this experiment on nodes reserved from the GENI Utah aggregate manager, then we run this same experiment on nodes reserved from the GENI Kentucky aggregate manager.

Figure 2 shows the number of control messages the new DIF member received during and after the enrollment procedure at Utah and Kentucky, respectively. The first few messages are for the enrollment procedure, and they include authentication messages, DIF configuration messages, and so on. After that, the number of messages continues to increase. This is because after the IPC process is enrolled into the DIF, the DIF Manager process continually sends probe messages to the IPC process, to check if the IPC process is still reachable. We observe similar results over the two aggregates, indicating that GENI supports repeatable experiments even across different aggregates.

B. Dynamic Layer (DIF) Instantiation

In this experiment, we demonstrate the dynamic layer instantiation feature of RINA by forming a new, higher level DIF layer. When two application processes try to establish a communication flow, but they cannot find a common existing

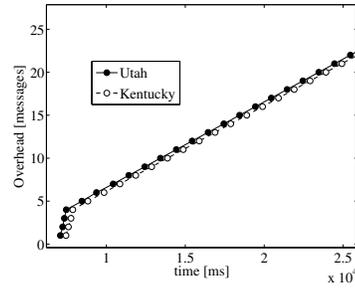


Fig. 2. Control messages received by the new DIF member: the first four messages are due to the enrollment procedure.

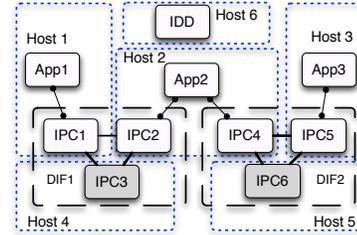


Fig. 3. Dynamic DIF Formation: App2 has underlying IPC processes in both DIF1 and DIF2, and uses the Inter-DIF Directory (IDD) mechanism to register a relay service between them.

DIF to communicate over, a new (higher level) DIF needs to be dynamically created.

We consider three application processes that use two low-level DIFs, DIF1 and DIF2 to communicate. Here DIF1 and DIF2 can be seen as two private local networks. Each underlying DIF has three IPC processes. In particular, App1 and App3 use IPC1 and IPC5 as underlying IPCs for their communication in DIF1 and DIF2, respectively, while App2 uses IPC2 and IPC4 for its communications in DIF1 and DIF2, respectively. IPC3 is the DIF manager of DIF1, and IPC6 is the DIF manager of DIF2 (Figure 3.)

App1 wants to establish a flow with App3, which can be seen as a process in one private network wanting to communicate with another process in another network. Since it cannot find a common underlying DIF, it triggers the Inter-DIF Directory (IDD) service mechanism and discovers which DIF it needs to join to establish a flow with App3. Previously registered as relay service for App3, the address of App2 is returned to App1. DIF3, is the new higher-level layer dynamically created by App2 (Figure 4.) The first member of DIF3 is IPC8, forked by App2, the DIF manager of the new DIF3. App2 then invites both App1 and App3 to have a new underlying IPC process to join and to enroll into the higher-level DIF3. Each application process forks a new IPC process (IPC7 and IPC9). As shown in Figure 4, DIF3 is dynamically formed, and App1 can establish a transport flow with App3 through DIF3.

Figure 5 shows the network topology displayed in Flack after we reserved six nodes using Omni from the Utah aggregate. Node PC (*pc121.emulab.net*), Node PC-0 (*pc116.emulab.net*), Node PC-1 (*pc99.emulab.net*)

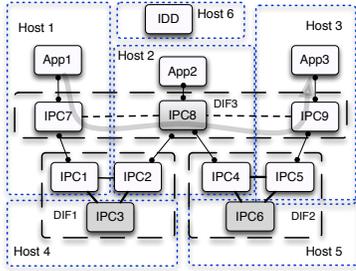


Fig. 4. Dynamic DIF Formation: higher-level DIF3 is dynamically formed by App2 upon App1's request to establish a flow with App3.

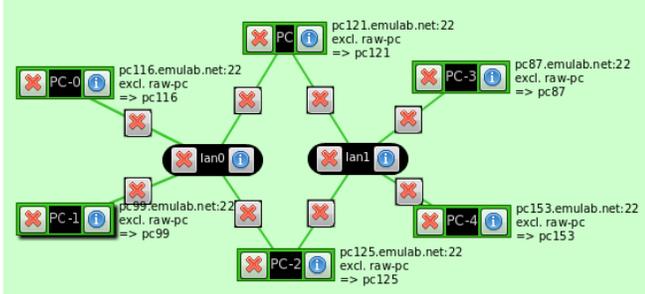


Fig. 5. Dynamic DIF Formation slice: network topology seen in Flack.

and Node PC-2 (*pc125.emulab.net*) are connected through LAN lan0. Node PC (*pc121.emulab.net*), Node PC-3 (*pc87.emulab.net*), Node PC-4 (*pc153.emulab.net*) and Node PC-2 (*pc125.emulab.net*) are connected through LAN lan1.

As shown in Figure 5, there are six GENI nodes corresponding to six hosts in Figures 3 and 4. The GENI Node PC is Host 6 which has the IDD process, Node PC-0 is Host 1 which initially has App1 and IPC1, Node PC-1 is Host 4 which has IPC3 (DIF manger of DIF1), PC-2 is Host 2 which initially has App2, IPC2 and IPC4, PC-4 is Host 5 which has IPC6 (DIF manager of DIF2), and PC-3 is Host 3 which initially has App3 and IPC5.

We run this experiment on resources from the Utah Aggregate and Kentucky Aggregate, respectively, and Figure 6 shows the number of keep-alive control messages received by App3 from App1 after the new higher-level DIF3 is formed and the transport flow between them is created.

VI. LESSONS LEARNED USING GENI AND CONCLUSIONS

We believe the deployment of future Internet architectures is an immense challenge. Having ours [15] or other prototypes constantly running on a GENI slice is key to enable users to test different competing architectures. In this paper, we reported on our experience, testing with our prototype, two key features of the RINA architecture [1], [2], [11], [12], [15]: enrollment of an IPC process into an existing DIF, and dynamic instantiation of a DIF layer, on top of two pre-existing DIFs.

We found that a slice with slivers across different aggregates, connected by a Generic Routing Encapsulation (GRE) tunnel could rarely be reserved. Moreover, we found

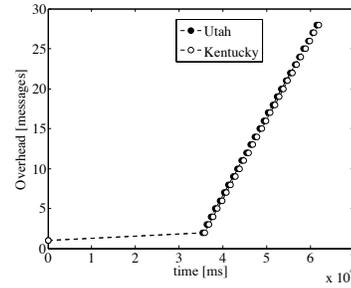


Fig. 6. Keep-alive control messages received by App3 from App1 once the new higher-level DIF3 is formed and the flow is created between App1 and App3.

it straightforward to reserve few slivers within a slice, but reserving a slice with complex topologies required several attempts.

Finally, although it was possible to run short-term experiments on the testbed, we also found it challenging to let users opt-in into a GENI slice hosting RINA, without having to renew (manually or automatically) the slice. In the future, we plan to expand our prototype evaluation, running it directly over layer-2 connectivity and testing the scalability of our architecture over a large-scale GENI slice.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation under grant CNS-0963974.

REFERENCES

- [1] J. Day, I. Matta and K. Mattar. "Networking is IPC: A Guiding Principle to a Better Internet". In Proc. of ReArch' 2008. Madrid, Spain.
- [2] J. Day. Patterns in Network Architecture: A Return to Fundamentals. Prentice Hall, 2008.
- [3] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. "Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Services". In Proc. of the ACM SIGCOMM 2012.
- [4] N. McKeown et. al. "OpenFlow: Enabling Innovation in Campus Networks". SIGCOMM CCR. vol. 38, no. 2, pp. 69-74, March 2008.
- [5] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, "Locator/ID Separation Protocol (LISP)". March 2009, draft-farinacci-lisp-12.txt.
- [6] GENI. <http://www.geni.net>.
- [7] ProtoGENI. <http://www.protogeni.net>.
- [8] B. Chun et. al. "Planetlab: An Overlay Testbed for Broad-coverage Services". SIGCOMM CCR. pp. 3-12, 2003.
- [9] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI Veritas: Realistic and Controlled Network Experimentation". In Proc. of the ACM SIGCOMM 2006.
- [10] B. White et. al. "An Integrated Experimental Environment for Distributed Systems and Networks". ACM SIGOPS Oper. Syst. Rev. 36, SI, December 2002.
- [11] E. Trouva, E. Grasa, J. Day, and S. Bunch. "Layer Discovery in RINA Networks". In IEEE CAMAD, 2012.
- [12] V. Ishakian, J. Akinwumi, F. Esposito, and I. Matta. "On Supporting Mobility and Multihoming in Recursive Internet Architectures". Comp. Comm., vol. 35, num.13 July 2012.
- [13] Pouzin Society. <http://www.pouzinsociety.org>.
- [14] Google Protocol Buffer. <http://code.google.com/p/protobuf>.
- [15] RINA Lab, Boston University. <http://csr.bu.edu/rina>.
- [16] Th. Megedanz and S. Wahle. "Control Framework Design for Future Internet Testbeds". Elektrotechnik und Informationstechnik. vol. 126, no.7, pp. 274-279. Springer, 2009.
- [17] ProtoGENI Flack. <http://www.protogeni.net/wiki/Flack>.
- [18] Omni. <http://trac.gpolab.bbn.com/gcf/wiki/Omni>.