

Experimenting with Routing Policies using ProtoRINA over GENI

Yuefeng Wang Ibrahim Matta Nabeel Akhtar
Computer Science Department, Boston University
Boston, MA 02215
{wyf, matta, nabeel}@bu.edu

Abstract—ProtoRINA is a user-space prototype of the Recursive InterNetwork Architecture (RINA), a new architecture that overcomes inherent weaknesses of the current Internet, *e.g.*, security, mobility, and manageability. By separating mechanisms and policies, RINA supports the programmability of different control and management policies over different communication scopes while using the same mechanisms. GENI (Global Environment for Network Innovations) provides a large-scale virtual network testbed that supports experimentation and possible deployment of future network architectures. In this paper, using ProtoRINA over GENI resources, we demonstrate how RINA’s support for the scoping of routing control and management, and instantiation of different routing policies, can be leveraged to yield faster convergence and lower routing overhead in the face of node or link failures.

I. INTRODUCTION

One fundamental problem of the current Internet is the lack of communication scopes. The whole Internet is treated as one single large network, which makes it extremely challenging to control and manage. Also the rudimentary “best-effort” delivery service of the Internet makes it even harder to support modern service requirements, *e.g.*, security, mobility, and new service models, *e.g.*, Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [1]. Many stopgap solutions have been proposed to address these challenges.

Our Recursive InterNetwork Architecture (RINA) [2], [3] is a new network architecture that is based on the fundamental principle that *networking is Inter-Process Communication (IPC)*. As a consequence, RINA inherently solves problems of the current Internet. RINA separates mechanisms and policies, and enables policy-based programming of user and management applications. And the recursive scoping of IPC services yields a better service management framework [4]. ProtoRINA [5] is Boston University’s user-space prototype of the RINA architecture. It can be used both as an experimental tool for developing new (non-IP based) applications and as a teaching tool for networking and distributed systems classes.

GENI (Global Environment for Network Innovations) [6] provides a large-scale experimental facility that supports experimentation with future network architectures for possible deployment. Using GENI, users can reserve computing resources from different locations that are layer-2 connected. In [7], we have demonstrated ProtoRINA over the GENI testbed through two fundamental RINA experiments: the enrollment of a process into a communication layer and the dynamic formation of such layer. In this paper we demonstrate, using ProtoRINA over GENI resources, how RINA’s support for the scoping of routing control and management, and instantiation of different routing policies, can be leveraged to

yield faster convergence and lower routing overhead in the face of node or link failures.

The rest of the paper is organized as follows. Our RINA architecture and ProtoRINA are briefly described in Section II and Section III, respectively. Our routing experiments and the GENI resources we used are described in Section IV and Section V, respectively. Experimental results are presented in Section VI. Section VII concludes the paper with future work.

II. RECURSIVE INTERNETWORK ARCHITECTURE

The Recursive InterNetwork Architecture (RINA) is a new network architecture that is based on the fundamental principle that *networking is Inter-Process Communication (IPC) and only IPC*. A *Distributed IPC Facility (DIF)* is a collection of distributed IPC processes with shared states, and a DIF provides communication service to application processes over a certain scope (*i.e.*, range of operation). Recursively, a DIF providing IPC service over a larger scope is formed based on lower level DIFs with smaller scopes. RINA separates mechanisms and policies, and IPC processes use the same mechanisms but may use different policies for different DIFs. For example, different DIFs can have their own routing policies. More generally, a set of application processes cooperating to perform a certain function constitute a *Distributed Application Facility (DAF)*. The function can be a communication service, weather forecast, genomics, *etc.* A DIF is a specialization of a DAF whose function is only to provide IPC service.

The Inter-DIF Directory (IDD) service is responsible for the naming management of RINA. It is used by a process to locate an application process (or service provided by a DAF). By querying the IDD, a service may be resolved either to an underlying DIF or to an IPC process inside a DIF through which the service can be accessed.

III. PROTORINA: A PROTOTYPE OF RINA

ProtoRINA [5], [8] is Boston University’s user-space prototype of the RINA architecture. Researchers can use ProtoRINA as an experimental tool to develop their own (non-IP based) user and management applications. Also, educators can use ProtoRINA as a teaching tool for networking and distributed systems courses to help students better understand basic concepts in the design of networks and applications. ProtoRINA has been tested on our local campus network, and we have also done some preliminary cross-debugging with other RINA prototypes (TRIA [9] and IRATI [10]). The current version is not a complete implementation of RINA and we continue to modify and add elements.

A. RINA Node

As shown in Figure 1, a *RINA node* is a host (or machine) where application processes and IPC processes reside. Application processes or high-level IPC processes communicate with their peers using the communication service provided by underlying low-level IPC processes, and the communication service is provided through an *IPC API*.

In our current implementation of ProtoRINA, TCP connections are used to emulate physical connectivity between IPC processes in level-0 DIFs. The emulation is provided by a shim layer, which includes a virtual link (wire) manager in each RINA node, and a RINA DNS server (connected to all RINA nodes) that is able to resolve a user defined level-0 IPC process' name to an IP address and port number.

Our current implementation supports a centralized IDD (responsible for naming management), and every RINA node is connected to it. The mapping between an application process (or high-level IPC process) to the lower level IPC process is done by the underlying DIF.

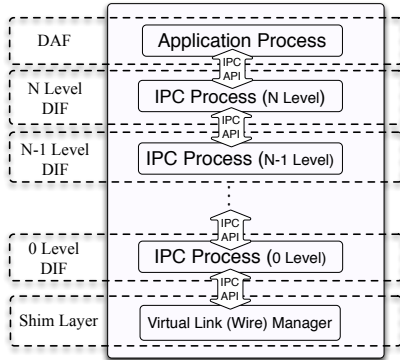


Fig. 1. RINA Node overview.

B. Configuration File

Each IPC process has a configuration file, which specifies IPC process' properties including name information, routing policies, underlying DIF information, *etc.* The RINA node and application process can also have their own configuration file.

The following are routing policies that are part of an IPC process' configuration file. In this example the IPC process is instantiated with a link-state routing protocol, link-state updates are sent to its neighbor processes every 10 seconds, and the path cost is calculated using hop count. ProtoRINA supports link-state routing and distance-vector routing.

```
rina.routing.protocol = linkState
rina.routingEntrySubUpdatePeriod = 10
rina.linkCost.policy = hop
```

Figure 2 shows the components of an IPC process and RINA APIs that enable the programmability of routing policies. Each IPC process has a Routing Daemon that is responsible for routing inside the DIF, and a Resource Information Base (RIB) that stores its view of all information related to managing the DIF. The RIB Daemon helps other components access information stored either in the local RIB through a *RIB API*, or in a remote IPC process' RIB by exchanging Common Distributed Application Protocol (CDAP) [3] messages with the remote IPC process. The IPC Resource Manager (IRM) is

responsible for managing the use of underlying IPC processes through an *IPC API* and for creating and managing connections between IPC processes. CDAP messages are sent over these connections through an *IRM API*.

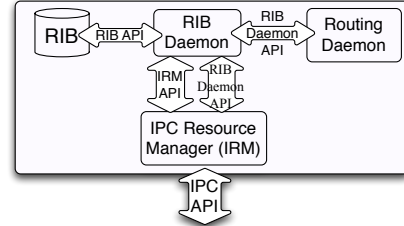


Fig. 2. IPC process' components and RINA APIs related to routing.

Based on a publish/subscribe model, the RIB Daemon provides timely information to the management (application) tasks of the DIF (DAF). Through the RIB Daemon API, the Routing Daemon can easily access the information needed for routing, such as link-state information from the IPC process' neighbors. As mentioned earlier, different routing policies can be set in the configuration file of the IPC process. Details of ProtoRINA components and RINA APIs can be found in [5].

IV. DESIGN OF ROUTING EXPERIMENTS

In our experiments, two application processes (App A and App B) on two different RINA nodes (Node 1 and Node 2) would like to communicate with each other. There are six other RINA nodes (Node A, Node B, Node C, Node D, Node E and Node F) which are able to provide relay service by building DIFs that include IPC processes on Node 1 and Node 2. We have a RINA node (Node DNS-IDD) running the RINA DNS server and IDD server. We have run experiments using two different DIF topology settings, but the same physical connectivity between RINA nodes.

A. Simple one-level DIF topology

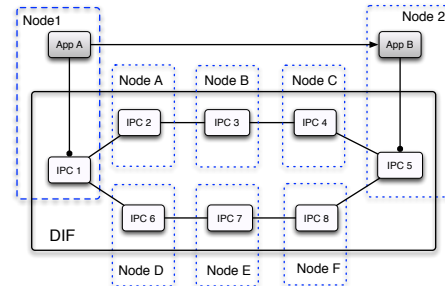


Fig. 3. App A and App B use a level-0 DIF consisting of eight IPC processes.

In this first setting (shown in Figure 3), we have only one large level-0 DIF consisting of eight IPC processes, and each IPC process resides on one RINA node. App A communicates with App B via the underlying level-0 DIF. App A and IPC 1 are on Node 1, and App A uses the communication service provided by IPC 1. App B and IPC 5 are on Node 2, and App B uses the communication service provided by IPC 5. For the other six RINA nodes, IPC 2 is on Node A, IPC 3 is on Node B, IPC 4 is on Node C, IPC 6 is on Node D, IPC 7 is on Node E, and IPC 8 is on Node F. Every RINA node is connected to the RINA node (Node DNS-IDD)

running the RINA DNS server and IDD server, which are not shown here.

First App A creates a connection to App B, and this connection is mapped to a flow in the underlying DIF from IPC 1 to IPC 5. Later App A starts sending messages to App B, where messages arrive at App B through the flow in the underlying DIF.

When an IPC process joins the DIF, it starts collecting the link-state routing information of the DIF. After a while the routing table of each IPC process converges. From IPC 1 to IPC 5 there are two paths with the same cost (using hop count as path cost), and we assume that at first IPC 1 picks the path IPC 1 - IPC 2 - IPC 3 - IPC 4 - IPC 5. So all messages from App A are sent over this path. After a while we shut down Node C. Through the link-state routing updates each IPC process soon learns of this change, and their routing tables converge again. The path from IPC 1 to IPC 5 adapts to IPC 1 - IPC 6 - IPC 7 - IPC 8 - IPC 5. During convergence of the routing tables, some messages sent from App A to App B get lost due to path failure. Once IPC 1 learns of the new path, the delivery of messages at App B resumes.

B. Two-level DIF topology

In this second setting (shown in Figure 4) we have four level-0 DIFs (DIF 1, DIF 2, DIF 3 and DIF 4), and one level-1 DIF (DIF 5). App A communicates with App B using the communication service provided by the underlying level-1 DIF (DIF 5), which recursively uses the communication service provided by level-0 DIFs. DIF 1 has three members IPC 1, IPC 2, and IPC 3. DIF 2 has three members IPC 4, IPC 5, and IPC 6. DIF 3 has three members IPC 7, IPC 8, and IPC 9. DIF 4 has three members IPC 10, IPC 11, and IPC 12. DIF 5 has four members IPC 13, IPC 14, IPC 15, and IPC 16.

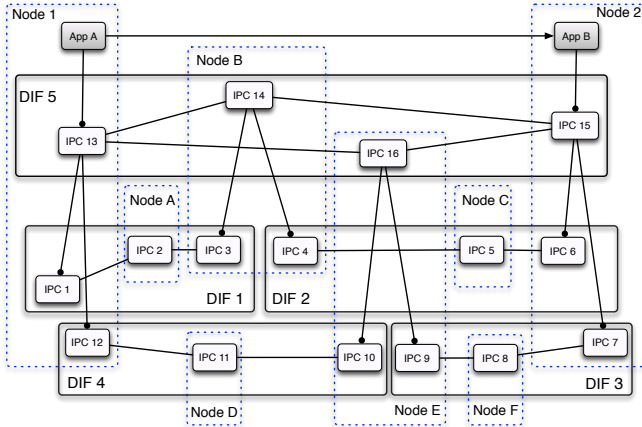


Fig. 4. App A and App B use a level-1 DIF DIF 5 which is built on top of four level-0 DIFs, and each level-0 DIF only consists of three IPC processes.

App A, IPC 1, IPC 12, and IPC 13 are on Node 1. App A uses the communication service provided by IPC 13, and recursively IPC 13 uses the communication service provided by IPC 1 and IPC 12. App B, IPC 6, IPC 7, and IPC 15 are on Node 2. App B uses the communication service provided by IPC 15, and recursively IPC 15 uses

the communication service provided by IPC 6 and IPC 7. IPC 2 is on Node A. IPC 3, IPC 4, and IPC 14 are on Node B, and IPC 14 uses the communication service provided by IPC 3 and IPC 4. IPC 5 is on Node C. IPC 11 is on Node D. IPC 9, IPC 10, and IPC 16 are on Node E, and IPC 16 uses the communication service provided by IPC 9 and IPC 10. IPC 8 is on Node F. Every RINA node is connected to the RINA node (Node DNS-IDD) running the RINA DNS server and IDD server, which are not shown here.

Similar to the previous setting, App A first creates a connection to App B, and this connection is mapped to a flow in the underlying DIF 5 from IPC 13 to IPC 15. Later App A starts sending messages to App B, and messages arrive at App B through the flow in the underlying DIF 5. From IPC 13 to IPC 15 there are two paths with the same cost (using hop count as path cost), and we assume that at first IPC 13 picks the path IPC 13 - IPC 14 - IPC 15. So all messages from App A will be sent over this path. When we shut down Node C, then DIF 2 cannot provide communication service between IPC 14 and IPC 15. This causes the (virtual) link between IPC 14 to IPC 15 to go down, which in turn makes the path IPC 13 - IPC 14 - IPC 15 unavailable. Through routing updates inside DIF 5, IPC 13 learns of this change and adapts to the new path IPC 13 - IPC 16 - IPC 15, then the delivery of messages at App B resumes.

V. RESERVING GENI RESOURCES

In our experiments, we reserve GENI resources through the GENI Portal [11]. The GENI Portal is a web tool that allows users to manage their projects and reserve GENI resources. Users can login to the GENI Portal using their home institution's credentials (username and password). After logging in, the user can use tools such as Flack and omni to reserve resources. We use Flack in our experiments.

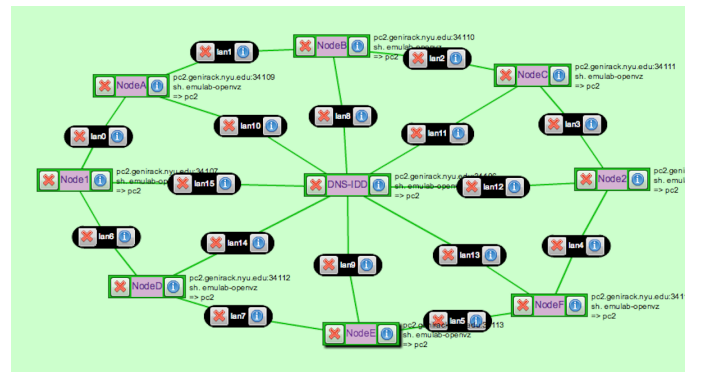


Fig. 5. GENI topology of nine virtual machines.

Figure 5 shows the network topology displayed in Flack after we reserve nine virtual machines (VMs) from the NYU aggregate. Each of the eight RINA nodes shown in Figure 3 or Figure 4 corresponds to a VM. We have the RINA DNS server and IDD server running on the ninth VM in the center of the topology, and this RINA node is connected to all RINA nodes to provide the directory service. After the VMs are reserved,

we SSH to each VM, upload our ProtoRINA code and run our experiments. Each RINA node generates a log file, and we obtain our experimental results by analyzing the log file.

VI. EXPERIMENTAL RESULTS

We run experiments on the GENI resources reserved in Section V with the two DIF topology settings described in Section IV. We try different routing policies by setting `rina.routingEntrySubUpdatePeriod` in the IPC process' configuration file to different values.

In the first setting, we have all eight IPC processes in the underlying DIF send link-state routing updates to their neighbor IPC processes every 10 seconds. In the second setting, in the level-0 DIFs (DIF 1, DIF 2, DIF 3 and DIF 4) we have all IPC processes send link-state updates to their neighbor IPC processes every 10 seconds. But in the level-1 DIF (DIF 5), we have all IPC processes send link-state updates to their neighbor IPC processes every 5 seconds.

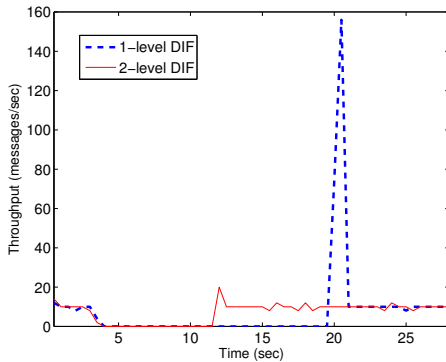


Fig. 6. Instantaneous throughput at App B.

App A sends messages to App B at a constant rate of 10 messages/second. In Figure 6, we show the instantaneous throughput starting from the time when App B receives the first message from App A. We can see that after Node C goes down, no new messages are delivered. Once the underlying flow between App A and App B adapts to another path, packet delivery resumes.

From Figure 6, we can see that in the second setting, routing recovers faster than that in the first setting. This shows that by building a higher level DIF with more frequent link-state updates on top of small level-0 DIFs with less frequent link-state updates, we can have a shorter recovery time.

We notice that message delivery exhibits a burst in a short time after recovery. This is because some messages enter in a routing loop between neighbor IPC processes whose routing tables have not yet converged. Once all routing tables converge, these messages get through to App B almost at the same time with more out-of-order packets delivered under the one-level DIF topology.

We also show the routing overhead at Node 2 over time in Figure 7 by measuring the amount (in bytes) of link-state packets received at Node 2. In the second two-level topology, this overhead is due to the total link-state update traffic received by IPC 6, IPC 7, and IPC 15 in their respective DIFs. We can see that these two settings have almost the same routing overhead at Node 2.

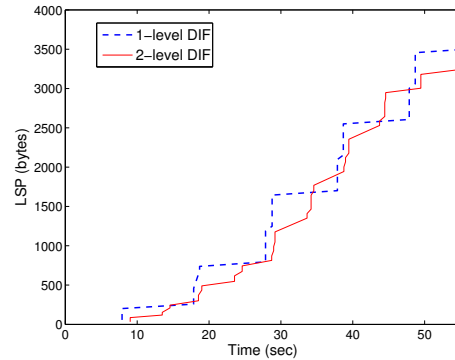


Fig. 7. Routing overhead at Node 2

From Figures 6 and 7, we can see that for a network with the same physical connectivity, by building DIFs using different topologies and different routing policies, we can achieve shorter recovery times without increasing routing overhead. We repeated our experiments on the NYU aggregate many times, and observed similar results.

VII. CONCLUSION AND FUTURE WORK

In this paper, we presented our experiments with different routing policies using ProtoRINA. The experimental results demonstrate that ProtoRINA provides a flexible platform for managing a network as it provides policy holders where users can set their own control and management policies.

We run our experiments on resources reserved from one aggregate at NYU, and as future work we plan to run experiments on GENI resources from multiple aggregates. Also we plan to set up a long-lived slice running ProtoRINA over the GENI testbed, to enable researchers and educators to opt-in and benefit from the RINA architecture by experimenting with policies for controlling and managing, not only routing, but also enrollment, authentication, and data transfer.

ACKNOWLEDGMENT

We would like to acknowledge the support of the National Science Foundation (NSF grant CNS-0963974) and the GENI Project Office.

REFERENCES

- [1] National Institute of Standards and Technology, "The NIST Definition of Cloud Computing," 2011.
- [2] Boston University RINA Lab, "http://csr.bu.edu/rina/."
- [3] J. Day, I. Matta, and K. Mattar, "Networking is IPC: A Guiding Principle to a Better Internet," in *Proceedings of ReArch'08 - Re-Architecting the Internet (co-located with CoNEXT)*, New York, NY, USA, 2008.
- [4] Y. Wang, F. Esposito, I. Matta, and J. Day, "RINA: An Architecture for Policy-Based Dynamic Service Management," in *Technical Report BUCS-TR-2013-014*, Boston University, 2013.
- [5] ProtoRINA, "http://csr.bu.edu/rina/protorina/."
- [6] GENI, "http://www.geni.net/."
- [7] Y. Wang, F. Esposito, and I. Matta, "Demonstrating RINA Using the GENI Testbed," in *Proceedings of the Second GENI Research and Educational Experiment Workshop (GREE2013)*, Salt Lake City, UT, USA, March 2013.
- [8] Y. Wang, F. Esposito, I. Matta, and J. Day, "Recursive InterNetworking Architecture (RINA) Boston University Prototype Programming Manual," in *Technical Report BUCS-TR-2013-013*, Boston University, 2013.
- [9] TRIA Network Systems, LLC, "http://www.trianetworksystems.com/."
- [10] The IRATI Project, "http://irati.edu/."
- [11] GENI Portal, "https://portal.geni.net/."