RINA: An Architecture for Policy-Based Dynamic Service Management

Yuefeng Wang Flavio Esposito Ibrahim Matta John Day

Computer Science Department, Boston University
Boston, MA 02215
{wyf, flavio, matta, day}@bu.edu

Technical Report BUCS-TR-2013-014

Abstract

Management is a vital component for delivering requested network services. The inability of the current Internet architecture to accommodate modern requirements has spurred attempts to provide novel network management solutions. Existing approaches often restrict the range of policies that can be employed to adapt to diverse network conditions. They are also tailored either to a specific set of (management) applications, or to the current Internet architecture, inheriting its shortcomings, for example adopting its incomplete (static) addressing architecture or ad-hoc solutions that result in so-called "layer violations."

In this paper, we describe a novel management architecture developed from first principles to enable the specification of various policies. To this end, we identify common underlying mechanisms, based on the unifying principle that any management application consists of processes maintaining and sharing distributed states of information objects for the purpose of delivering a network service. This principle underlies our Recursive InterNetwork Architecture (RINA), where the notion of a "layer" represents such state sharing among processes and such layers can be repeated over different scopes and stacked to provide more effective wide-area services. We present a management framework that enables application programmers to specify four types of policies for managing a layer, the network, the naming of services, and an application, and use our prototype implementation to demonstrate adaptation to changing communication and load requirements.

Categories and Subject Descriptors C.2.1 [*Network Architecture and Design*]: Distributed networks

General Terms Management, Design

Keywords Network management, Network protocols

1. Introduction

The lack of a complete architecture continues to dramatically increase the management costs and complexity of all distributed systems that are based on a service paradigm. Service management today must support a collection of protocols with different scalability and convergence properties, each tailored to a specific management application. [1]

Millions of dollars have been spent in the last five years on middlebox hardware and maintenance [32]. In response to modern requirements, *e.g.* security or mobility, network and service providers as well as enterprises have adopted ad hoc solutions, and so-called "layer violations", where passing datagrams are deeply inspected so as to perform application- or transport-specific processing. In-network solutions are often complex (as specific modules need to be installed and maintained with patches), are limited to forwarding in IP devices (see *e.g.* OpenFlow [24]), and lack a complete management application architecture.

The desire to simplify and better manage a distributed system in support of modern service requirements and new revenue models, together with the inability of the current Internet to cope with the modern requirements has recently fostered research in novel service-based architectures (see *e.g.* [10, 19, 26]), and extensible management control platforms (see *e.g.* [11, 20, 31].) Some of these solutions target specific problems of the current Internet, *e.g.* mobility or multihoming, while others focus on the manageability of distributed systems [15, 20] or their scalability [11]; yet other solutions focus on the expressiveness of adaptive-based protocols that enable systems to tune behaviors of management applications by policy instantiation [21].

We generalize existing work on service management to have a complete and general-purpose architecture that includes the minimal set of mechanisms serving different and scalable management goals, each implementable with the simplest possible control platform. In particular, we design and implement an API for extensible, policy-based network service management that is (*i*) transparent, *i.e.* able to hide

[1] We extend the definition of a management application given as "*the logic needed to control management features, e.g. routing or access control*" [20] by including also the set of processes that share states to perform such features.

the implementation complexity of mechanisms, (*ii*) minimal but complete, (*iii*) dynamic, *i.e.* supports policy adaptation as network state changes, and (*iv*) general, *i.e.* usable by a large set of management applications at different time scale and across different scopes. The API is provided within the framework of our Recursive InterNetworking Architecture (RINA), whose design and implementation did not arise *de novo*, but instead derives from first principles [4, 5].

RINA is based on the fundamental principle that networking is inter-process communication (IPC.) It recurses the IPC service over different scopes. IPC processes are merely application processes that are members of a Distributed IPC Facility (DIF.) Specifically, a scope defines a 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. Application processes can themselves be IPC processes of an upper DIF that is providing services over a wider scope.

A DIF is an organizing structure — what we generally refer to as a "layer." A DIF is a collection of IPC processes (nodes.) Each IPC process executes routing, transport and management functions. IPC processes communicate and share state information. How a DIF is managed, including addressing, is hidden from the applications. We extend the notion of a DIF to a Distributed Application Facility (DAF). A DAF does not support IPC but includes management functionalities. [2]

We introduce the management architecture of a DAF, identifying four forms of management within the RINA design principles: layer, network, naming and application (Section 2.) We then present our Resource Information Base (RIB)-based policy specification language (Section 3.) Based on a pub/sub model, the RIB-based management mechanisms and interfaces include a distributed set of RIBs and RIB daemons (processes) that together with a common application protocol (CDAP), update and maintain the state of management objects shared among the processes composing the DIF or DAF.

We evaluate with our prototype few representative examples of how our platform enables each of the four management functionalities across different styles of management ranging from totally decentralized, *i.e.* autonomic, to centralized, *i.e.* manager-agents style, to hybrid approaches, *e.g.* hierarchical (Sections 4 and 5.)

Before concluding our work (Section 7), we also show how our contributions differ from related service-based and network management architectures, as well as from other extensible control platforms and recent declarative-based solutions (Section 6.)

[2] A DAF is logically built on a DIF that supports its inter-process communication. A DIF is a particular case of DAF that only provides communication.

2. Recursive InterNetwork Architecture

The basic premise of this architecture, yet fresh perspective, is that networking is not a layered set of different functions but rather a single layer of distributed Inter-Process Communication (IPC) that repeats over different scopes. Each instance of this repeating IPC layer implements the same functions (mechanisms) but policies are tuned to operate over different ranges of the performance space (*e.g.*, capacity, delay, loss.) In this model, application processes communicate via an IPC facility and they can themselves be IPC processes for higher level IPC facility (hence the recursion.) Since the facility is distributed, we call it a Distributed Inter-process communication Facility (DIF).

DIFs are different from the traditional definition of layer in the Transmission Control Protocol/Internet Protocol (TCP/IP) architecture. First, they perform a coordinated set of policy managed mechanisms to achieve the desired (IPC) service rather than a single mechanism or small subset of predetermined mechanisms. Second, a DIF naturally separates various concerns, including operation over different time scales (*e.g.* short-term data transfer and multiplexing vs. long-term connection management and access control issues.)

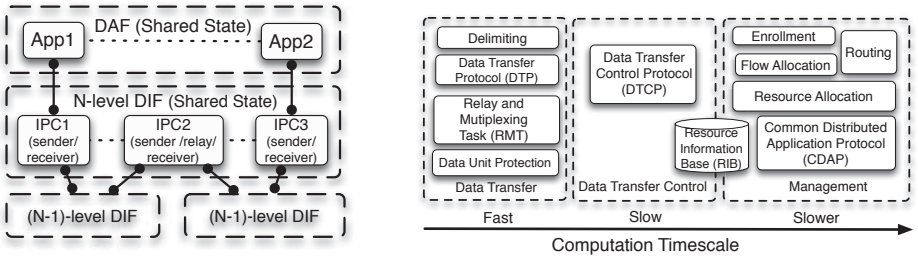
More generally, we call a set of application processes cooperating to perform a certain function a Distributed Application Facility (DAF.) We focus on particular DAFs that are used for network service management. Each member within the DAF is called Distributed Application Process (DAP.) A DIF is a specific DAF whose mechanisms are limited to communication (Figure 1*a*.)

We elaborate on these concepts in the next subsections, where we provide a more detailed overview of the RINA design principles and components (Section 2.2) within the management architecture framework of a distributed application facility (Section 2.1.)

2.1 Management Architecture and Design Principles

The Distributed Application Facility (DAF) management architecture consists of four forms of management: (*i*) *layer/DIF management* for the purpose of managing the DIF itself so as it provides IPC (*e.g.*, assigning addresses to IPC processes and routing among them); (*ii*) *network management* for the purpose of managing the various DIFs that make up the whole network (*e.g.*, the dynamic creation of new, wider-scope, DIFs to enable communication between remote application processes); (*iii*) *naming management* for the purpose of maintaining service/application names and resolving them to addresses of IPC processes through which services can be reached, and (*iv*) *application management* for the purpose of supporting specific applications, *e.g.* file sharing or content distribution.

In the rest of this section we describe the RINA design principles, and how they relate to each management architecture domain. First, we introduce the three *independent*



(*a*) A RINA network (*b*) RINA mechanisms within an IPC process

Figure 1. (a) Example of a RINA network: RINA has one building block (the DIF layer) that can be composed to form a higher-level DIF from the services provided by lower-level DIFs. This figure illustrates composition by recursion of the DIF layer. (b) Architecture of an IPC: each IPC process consists of three distinct sets of mechanisms dealing with IPC aspects at different time scales: IPC data transfer, IPC data control and IPC management.

principles — the first two relate to layer/DIF management and the third to network management, and later we move our attention to the two *dependent* principles, the first relates to naming and the second to application management.

Networking is inter-process communication. The first and foremost design principle of the architecture is based on the redefinition of a layer. Following R. Metcalfe's quotation (1970) "*networking is Inter-Process Communication*", we define a layer as a distributed IPC facility, that recursively repeats over different scopes. This means that a RINA network is not a layered set of different mechanisms, but all layers have the same building block, an IPC process. A distributed set of application processes, *e.g.*, a network management application, use the IPC facilities to share and modify network states (Figure 1a.)

All layers have only three sets of mechanisms, repeated over different scopes. Each IPC process consists of three sets of mechanisms dealing with communication aspects at different time-scales: (i) a set of fast (frequently called) mechanisms for IPC *data transfer*, *e.g.* Service Data Unit (SDU) fragmentation, (ii) a set of slower mechanisms for IPC *data control*, *e.g.* flow control, and (iii) the set of IPC *management* mechanisms, *e.g.* routing, that operate at an even slower rate, *i.e.* infrequently called (Figure 1b.) 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 operate over longer time scales.

DIFs can be dynamically instantiated. Management applications adapt (network) configurations by modifying or instantiating policies. Policies are tuned transparently, *i.e.* hiding the implementation complexity of mechanisms, and dynamically, *i.e.* without configuration transit anomalies or service interruptions. The conditions of a managed network may in fact change over time: the level of congestion may increase, a denial-of-service attack may occur, or an application instance may migrate for maintenance or change its point of attachment while moving. RINA provides an abstract interface to programmers of any management application to support such configuration changes.

The above three *independent* design principles lead to more specific *dependent* principles. We describe them, starting from one, direct consequence of the *dynamic DIF instantiation* that solves the mobility and multihoming problem (*i.e.*, having more than one connection to the network).

Dynamic late binding. As virtual memory hides the allocation of physical memory – thus alleviating applications from the burden and overhead of memory management – RINA makes management application programming (*e.g.* routing) easier by dynamically binding service (application) names to addresses. An application process in RINA requests communication with another application process from its underlying DIF using the name of the destination application. The name is resolved by the DIF to the address of the IPC process through which the destination application can be reached. If the request is accepted, a flow instance is allocated and dynamically assigned a connection end-point identifier (CEP-id). The application process is also dynamically allocated a flow handle (port number) that is mapped to the CEP-id. Unlike TCP/IP, where applications are privy to addresses and well-known ports, RINA addresses inside a DIF are invisible to the application processes — they are privately assigned to IPC processes when they join the DIF (and are authenticated). RINA addresses also specify processes in the same DIF level, unlike TCP/IP where IP addresses are bound to (lower-level) interface addresses. Addresses in RINA are relative, i.e. the address of a process at one DIF level is viewed as a name by a lower level DIF.

Separation of mechanisms from policies. A policy is a variant aspect of a mechanism. For example, acknowledgment is a mechanism; when to acknowledge is a policy. By only instantiating policies through the given interfaces (Section 3.4), the complexity of management applications is reduced. One important implication of this principle is that *each DIF or DAF can have its own policies*. This principle has several merits; we only discuss one for its historic importance. It is known that, when managing a large scale network, a connectionless approach is preferable at the edge (where frequent loss or corruption occurs) for its resilience, but it is prone to congestion, while connection-oriented networks are preferable in the core as they avoid congestion even though they require connections to be re-established or supported by redundancy as they are fragile to failure. By instantiating the appropriate policies, the advantages of connectionless and connection-oriented networking can coexist without their shortcomings. More generally any management policy can be instantiated in combinations with different scopes.

2.2 IPC Mechanisms

In this section we describe the architectural elements of an IPC process and the mechanisms required for communication between any (management) application processes using an underlying IPC facility.

Application processes establish communication using their underlying IPC facilities. To provide such communication service, IPC processes employ three types of mechanisms (Figure 1b): (*i*) *IPC data transfer*, that includes data unit protection and delimiting mechanisms, a Data Transfer Protocol (DTP), and a Relay and Multiplexing Task (RMT) to efficiently share the IPC facility among several flows. (*ii*) *IPC transfer control*, that supports requested channel properties during data transfer via a Data Transfer Control Protocol (DTCP); we name the combination of DTP and DTCP as Error and Flow Control Protocol (EFCP.) (*iii*) *IPC management*, that handles the functionalities needed to establish and maintain states, and includes a Common Distributed Application Protocol (CDAP) to operate on, or populate a Resource Information Base (RIB) with states, *e.g.* application names, addresses, and performance capabilities used by various DIF management tasks, such as routing and flow management.

Although we implemented the relay and multiplexing mechanism, as well as the basic functionality of the Data Transfer Protocol for flow management, the focus of this paper is on the design and implementation of the IPC management mechanisms. [3]

[3] The implementation of a fully-fledged efficient Error and Flow Control Protocol is part of our ongoing and future work.

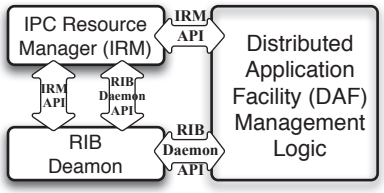


Figure 2. Using our policy specification language, Distributed Application Process (DAP) programmers can specify the logic for managing a layer, the network, the naming of services, or an application.

3. RIB-Based Management

Transparency is the ability of hiding the complexity of the implementation of mechanisms of a (distributed) system from both users (of a service or applications) and application programmers. To provide transparency, a distributed system architecture should offer interfaces to the (physical, virtual or logical) resources, so that such resources *appear* to be locally available. An object model is the means by which such transparency is provided, and consists of a set of interfaces to the objects (resources), a set of (CDAP) operations on the objects, and a broker or *RIB daemon* to handle such operations.

As the Management Information Base (MIB) defined in [18] or the Network Information Base (NIB) defined in Onix [20], our Resource Information Base (RIB) is a partially replicated distributed object-oriented database that contains the union of all managed objects within a DAF, together with their attributes. The role and responsibilities of the RIB daemon are similar to those of memory management in an operating system: to manage the information stored in the RIB and its veracity, updating and making states available to (IPC and application) processes.

Each form of management — layer, network, naming and application (Section 2.1) — is performed by a Distributed Application Facility (DAF). Based on a pub/sub model, a distributed set of RIBs and RIB daemons enable Distributed Application Processes (DAPs) to specify different styles of management within each form, ranging from fully decentralized, *i.e.* autonomic, to centralized, *i.e.* manager-agents style, to hybrid approaches, *e.g.* hierarchical. Each (management) DAP is composed of an IPC Resource Manager (IRM), a RIB daemon, and a user-defined component that implements the logic of the distributed management application facility using our API as a policy specification language (Figure 2.) The DAF responsible for layer management is composed of the management part within each IPC process (Figure 1b). The network management DAF is handled by the Network Management System (NMS) of each DIF. The Inter-DIF Directory (IDD) service is responsible for naming management by answering queries via a set of local agents responsible for intra-DIF name resolution.

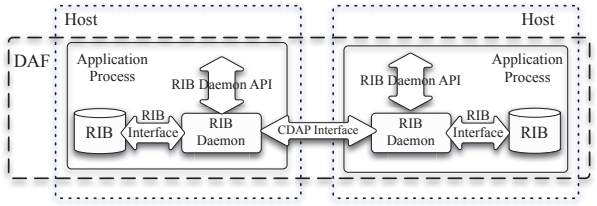


Figure 3. The RIB daemon is responsible for managing the information stored in the RIB and its veracity, updating and making states available to (IPC and application) processes.

In this section we describe the broker architecture (Section 3.1), the CDAP operations (Section 3.2), the IRM component (Section 3.3), and the RINA API (Section 3.4.)

3.1 Broker (RIB Daemon)

Similarly to traditional existing object models [12, 28], our architecture has a broker (part of the RIB management) responsible for allowing management processes to transparently make requests and receive responses. The broker handles such communication with *subscription* events. A subscription represents the quantity and type of information, to propagate in predefined situations by a management application on objects when specific situations occur.

Subscription events are mapped by the broker that recognizes the objects from their type, and acts upon different requests with a set of operations on objects in a local or remote RIB on behalf of the application process. Subscriptions have equivalent design goals as the *notification* events defined by the OSI model [15], or *traps* in the Simple Network Management Protocol (SNMP) [3], though they support more advanced control operations. For example, as in [11], they allow efficient control operations by supporting the separation of local frequent management operations from wide-area and infrequent operations. A local (read / write) operation may consist of a direct access to attributes of objects in a local RIB, *e.g.* for link (or any resource) discovery, while an infrequent and wide-area operation on remote objects may be routing updates, by generating a sequence of CDAP messages on remote RIBs.

3.2 Operations on Objects: Common Distributed Application Protocol (CDAP)

To share or modify states such as routing updates, and to coordinate joint operations on communication instances in a distributed management application, RINA defines a set of operations executable on (remote) objects. Such operations are supported by a Common Distributed Application Protocol (CDAP) whose design is based on modular, object-oriented principles [15] [13]. CDAP is based on three logical modules: a Common Application Connection Establishment (CACE) module, required for application protocol and syntax agreement within the application connection, an (optional) authentication module, and a set of CDAP messages for operating on objects.

With the goal of designing a general management (application) protocol, CDAP is responsible for disseminating the information necessary for the coordination of any DIF or DAF. To this end, we observe that only six operations can be performed on objects — create/delete, read/write, and start/stop — and that only two operations — connect and release — are sufficient to enable authentication and coordination among instances (entities) of a management application.

While application processes are free to define new types of objects to serve any (management) purpose, as long as all the application instances agree on the same object data representation, we define the DAF management mechanisms such that they impose restrictions on specific RINA uses of CDAP, *i.e.* on how to form and manage a distributed IPC facility with the enrollment procedure (Section 5.1) and other management operations, *e.g.* neighbor discovery.

3.3 IPC Resource Manager (IRM)

The IPC Resource Manager (IRM) is an important component of the architecture, present in all application processes. It is responsible for managing the use of underlying DIFs and their IPC processes, including the allocation and deallocation of flows among such processes.

To establish a communication, application instances need to allocate a flow. Such flow requests, initiated by the application using its IRM API (Table 1), are handled by the IRM and forwarded to one of its underlying IPC processes that can provide transportation service to the desired destination by using its IPC Flow interface.

The flow allocator of the underlying IPC process forwards the allocation request to the remote application process. If the request is accepted, a flow identifier is generated and returned to the requesting application process (Figure 4) to use for sending/receiving data.

3.4 RINA API

We design and provide a *general* API that simplifies the design and development of sophisticated and scalable management applications, leveraging the separation between mechanisms and policies, thus allowing (management) application processes to read and write any state — set of object attributes — of any manageable element of the network (*e.g.* a router.)

Object attributes can be read or written through a *general* subscription mechanism that includes registration for passive (subscribe) or active (publish) notifications of local or remote state changes. In support of subscription generation events, we provide a set of flow management IPC Resource Manager (IRM) API, *i.e.* for allocation, deallocation and usage of a flow (Table 1.)

Every application instance has a copy of its management states stored in a distributed data structure that we call *Resource* Information Base (RIB). Every (IPC and application)

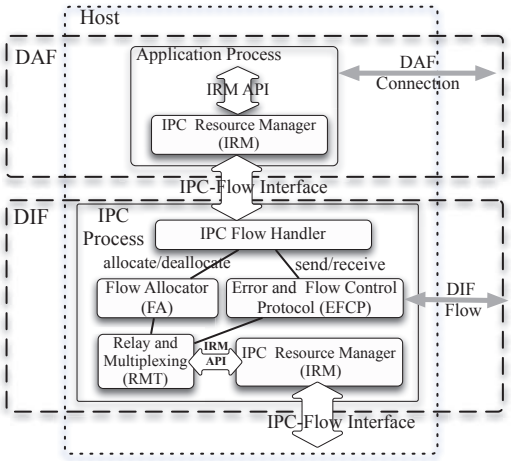


Figure 4. The IPC Resource Manager (IRM) is responsible for managing the IPC processes underlying the application, including the allocation and deallocation of flows.

RIB Daemon API		
Name	**Requires**	**Returns**
createSub	*AttributeName Publisher Frequency*	*SubID*
deleteSub	*SubID*	–
readSub	*SubID*	*Attribute*
createPub	*AttributeName Frequency*	*PubID*
deletePub	*PubID*	–
writePub	*PubID Attribute*	–
IPC Resource Manager (IRM) API		
Name	**Requires**	**Returns**
allocateFlow	*SrcAppName DestAppName*	*FlowID*
deallocateFlow	*FlowID*	–
send	*FlowID Message*	–
receive	*FlowID*	*Message*

Table 1. RINA API: The RIB Daemon API is used for read or write operations on members and attributes of the RIB via publish or subscribe events. The IPC Resource Manager API is used to manage flows between application processes.

process has a `Broker` (or RIB daemon), responsible for managing the subscriptions and updating the RIB distributed object database. The RIB is the core of our service management architecture and represents a generalized case of the *Routing* Information Base stored in IP routers. Rather than only storing prefixes to destinations, our RIB stores all the resources accessible by the management application instance.

When a management process (for example a routing application process) subscribes to a service, a subscription event is created. A subscription requires a set of parameters: (*i*) a mandatory unique subscription ID, limited to the scope of the service; (*ii*) a list of attributes to be exchanged, (*iii*) a list of members among which the attributes are exchanged, (*iv*) a logical expression that defines the time scale at which those information shall be exchanged with some (*v*) variance or tolerance, and (*vi*) a flag on whether the process needs to read or write (*i.e.* subscribe or publish) the attributes.

The RIB subscription mechanism is a generalized case of every publish-subscribe paradigm. Standard publish-subscribe systems are usually asymmetric. The RIB subscription mechanism not only supports both the symmetric and asymmetric paradigms, that is, the publisher process is capable of selecting the subscribers, [4] but also the traditional query-based paradigm is supported, where a process sends a request to a server and waits for its response.

Note also while the RIB is the means by which management applications are scalable and resilient to failure, RINA relies on the logic of the management application to achieve such desirable properties; (partial) copies of a RIB may be in fact replicated and distributed among multiple application instances if configured by the management application.

Example *(subscription and API usage)*. As an example of our policy language specification within our RIB-based management framework, we walk through the sequence of steps behind a simple subscription. We show how an application process `AP1` obtains a link state information from its direct neighbors `N1` every 10 seconds for routing purposes. `AP1` uses its RIB daemon API to subscribe to `N1` link state:

```
createSub("linkState", N1, 10).
```

The subscription is handled by the RIB daemon and converted into a CDAP message requesting the link state over a flow, previously allocated by using its IRM API:

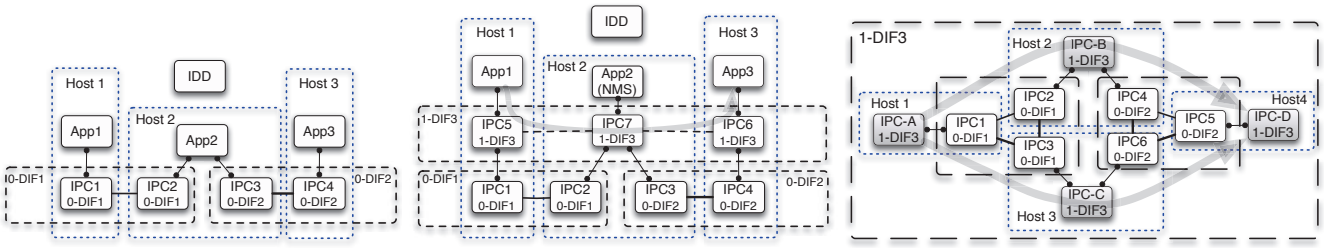
```
flowID  =  allocateFlow(AP1, N1)
send(flowID, CDAPMsg).
```

4. Prototype Implementation

This paper uses initial version of our RINA prototype, which consists of 28,000 lines of Java code, excluding support libraries, test applications, and configuration files, and it was tested with application and IPC processes running on the local campus network, both on Linux machines and on Android phones. Also we have demonstrated our prototype on the GENI testbed [35]. The latest version of our RINA prototype can be found at [29].

In all our experimental settings, we use machines with an Intel Core 2 Quad CPU at 2.66 GHz and 4 GB of memory, running Scientific Linux 6.2 (Carbon). The implementation of the data transfer mechanisms is limited to the relay, multiplexing and flow management within the Data Transfer

[4] For example a management application sending routing updates may prefer to be (temporarily) silent with subscribers along a given path, because of congestion, or because it has detected a misconfiguration or a suspicious behavior.



(*a*) App2 offers a relay service (*b*) 1-DIF3 is dynamically formed (*c*) path switching

Figure 5. (a) Naming management: App2 has underlying IPC processes in both DIF1 and DIF2, and uses the Inter-DIF Directory mechanism to register a relay service among them. (b) Network management case study: 1-level DIF is dynamically formed by App2 upon App1 request to establish a flow with App3. (c) Layer/DIF management case study: processes involved in the routing management application, built to show how in RINA different routing policies can be easily instantiated to improve performance.

Protocol (DTP). An efficient kernel implementation of such and other shorter time-scale data transfer mechanisms is deferred to another paper as we focus here on longer timescale management issues.

All management (CDAP) and transfer (DTP) messages are serialized and deserialized using the Google Protocol Buffers [9]. The prototype is built as an overlay over the Java TCP library, where TCP connections are used to emulate physical connectivity.

To demonstrate how RINA enables policy-based dynamic service composition, we built few example applications that illustrate its use in managing a distributed facility (Section 5.)

5. Service Management Case Studies

In this section we show how the RINA API is used as a policy specification language to build a set of management applications. We present various case studies to demonstrate the different forms of management:

(1) Layer/DIF management — we use the proposed API to trigger an enrollment procedure (Section 5.1), and to specify a link-state routing policy in which each IPC process of a DIF publishes its link state, at a certain frequency, and subscribes to the link state of every other IPC process. The latter is an "autonomic" style of management within the DIF where each IPC process is configured to independently respond to changes in the network state (Section 5.3.)

(2) Network management — for application processes to communicate, we show a case where they lack an underlying DIF that can support their IPC communication. Thus they need to join a new wider-scope (higher-level) DIF initiated by a relay application process. The source and destination application processes enroll into the new DIF subject to the management policies of the new layer/DIF (Section 5.2.) This is a centralized style of management where the relay application acts as the entity controlling the enrollment into the new DIF.

(3) Naming management — we present the case of an application process requesting the allocation of a communication flow through its underlying DIF using the service/application name (Section 5.2.) If the service is reachable through an underlying DIF, then a flow handle associated with the destination IPC process hosting the service is returned. Otherwise, if the service is unreachable, then a wider-scope (higher-level) DIF needs to be created. In this latter case, the service name is resolved to an application (relay service) that initiates the creation of a new DIF (see (2)) through which the service can be reached. This is a hierarchical style of management where underlying DIFs first try to locally resolve the service name, before a global name resolution takes place.

(4) Application management — we used the proposed API to specify a load balancing policy where load balancing is done hierarchically by distributing requests across two DIFs, both hosting the service, then over the chosen DIF, the service name is registered to the application process that is least loaded and that mapping is published to the directory service (Section 5.4.)

The above common management applications were chosen as they are usually statically or manually configured, *i.e.* unable to adapt to external state changes, but they potentially impact system performance, such as decreasing throughput and increasing delay. In our description, we point out the mechanisms and the policies being used, and we show few experimental results.

5.1 Layer Management Case Study: Enrollment Procedure

As an example of a Distributed Application Facility (DAF) with layer (DIF) management functionality, we describe the 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. [5]

In RINA, IPC processes need to be enrolled into the same DIF to communicate. The procedure starts with a subscription call by the enrollment DAP, using the RIB daemon interface (Section 3.4):

```
DIFSubID = createSub(DIFName, IDD, - )
```

The subscription has a single object attribute named `DIFName`, and triggers a query to an Inter-DIF Directory (IDD). The query contains such DIF name attribute in which the DAP needs to enroll. The directory service mechanism dictates that the query reply must contain at least the address of an IPC process within the requested DIF (if any), and other optional information such as a list of supporting DIFs that can be used to reach the IPC process, or any DIF policy (for example their authentication.) Once the DAP has obtained the address of such IPC process, it requests authentication (*e.g.* with user and password or SSH keys.) If successfully authenticated, the IPC process assigns the new member a name — an 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 IPC process that enrolled the new DIF member updates all the other existing IPC processes with the new member information, such as new application processes reachable from it.

5.2 Network Management Case Study: Dynamic DIF Formation

As an example of DAF with network management functionality, we walk through the steps necessary to dynamically form a DIF. In this case study, the DAF is simply the Network Management System (NMS) process managing the new DIF. The enrollment procedure allows IPC communication and the formation of a distributed IPC facility (DIF.) Management application processes can then share and modify their states within that facility to manage the DIF so as to provide communication for application processes. In this subsection we consider two application instances trying to establish a communication flow even though they cannot find a common existing DIF to communicate over: a new (higher level) DIF needs to be dynamically created.

We consider three application processes that use two 0-level DIFs, DIF1 and DIF2 to communicate (Figure 5a.) Each underlying facility has two IPC processes. In particular, `App1` and `App3` use IPC1 and IPC4 as underlying IPCs for their communication in DIF1 and DIF2, respectively, while `App2` uses IPC2 and IPC3 for its communications in DIF1 and DIF2, respectively.

[5] Examples of enrollment procedures in today's Internet are found in the IEEE 802.11 and IEEE 802.1Q. Other examples are the assignment of MAC addresses, or the creation of a listening well-known socket (see [4], Ch. 2.)

`App1` creates the subscription whose object contains a single attribute, the remote application instance name `App3`:

```
DIFSubID = createSub(applicationName, IDD, - )
```

to trigger the Inter-DIF Directory service mechanism and discover which DIF it needs to join to establish a flow with `App3`. If previously registered as relay service for `App3`, the address of `App2` is returned to `App1`. DIF3, is the new 1-level facility dynamically created by `App2`. The first member of DIF3 is IPC7, forked by `App2`, manager of the new DIF3. `App2` then invites both `App1` and `App3` to join and to enroll into the 1-level DIF3. Each application process forks a new IPC process (IPC5 and IPC6.) Now that DIF3 is dynamically formed, `App1` can establish a connection with `App3` through DIF3.

5.3 Layer/DIF Management Case Study: Policy-Based Routing

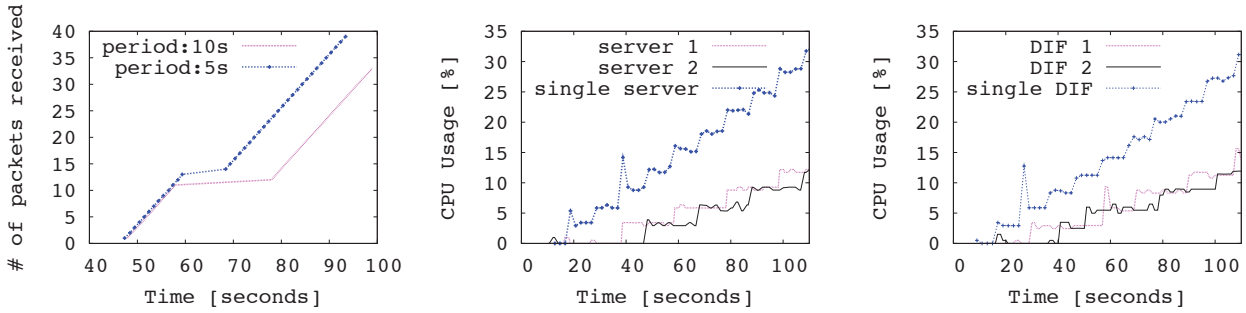
Due to a wide range of variability in network connectivity and also a wide range of data traffic patterns, a *one-size fits-all* routing protocol does not exist. As previously studied [21], even hybrid routing protocols (*e.g.* [30]) perform well only under certain conditions and require additional heuristics to achieve good performance in scenarios for which they were not designed. Policy-based management protocols instead, such as routing, enable systems to promptly adapt their behavior to different (management) applications as well as to external conditions, *e.g.* to virtual server migrations or failure. In this section we show how in RINA, a routing DAF enables better data delivery performance with respect to its alternative statically configured solution by allowing flexible policy specification.

Procedure 1 : `Link State Routing`

```
1: NbrSubID = createSub('neighbour', myself, f1)
2: SelfLSSubID=createSub('linkState', myself, f2)
3: LSPubID=createPub('linkState', f3)
4: NbrList = readSub(NbrSubID)
5: LSSubID=createSub('linkState', NbrList, f4)
6: repeat every T seconds
7:    SelfLinkState = readSub(SelfLSSubID)
8:    writePub(LSPubID, SelfLinkState)
9:    GlobaLinkState = SelfLinkState
10:   NbrLinkState = readSub(LSSubID)
11:   GlobaLinkState=GlobaLinkState ∪ NbrLinkState
12:   FWTable = buildFWTable(GlobaLinkState)
13: end repeat
```

Using Dijkstra's algorithm, we implemented a link state routing as a DIF management application. Consider Procedure 1. To compute a routing table, a map of the network connectivity needs to be determined by distributing the cost to reach each neighbor. [6] To this end, each (routing) application process initiates a discovery mechanism to broadcast

[6] We used the end-to-end delay as a cost metric policy.



(*a*) Path switching (*b*) Intra-DIF load balancing (*c*) Inter-DIF load balancing

Figure 6. (a) Throughput improves due to a faster path adaptation after a main path failure under more frequent link state updates. (b) Intra-DIF load balancing: Lower CPU usage is measured on each server when the intra-DIF load balancing policy is applied within the same facility. (c) Inter-DIF load balancing: lower CPU usage is measured on each server when the IDD load balancing policy is applied across facilities.

its link state, and obtain the link states of its neighbors (Procedure 1, lines $1-5$.) In particular, each routing application process subscribes to its own RIB Daemon to learn of its available neighbors (line 1) and cost to reach them (line 2), and then publishes its own link state (line 3 and lines $7-9$.) It also creates subscriptions to all its neighbors to get their link states (lines 4 and 5.)

Such link state information obtained from its neighbors (line 10) together with its own link state information (line 9) is later (recursively) aggregated (line 11) to compute the network connectivity map (denoted as *GlobaLinkState*) and to build the forwarding table (line 12.) Lastly, lines $7-12$ are repeated for the routing application to adapt to possible changes of the network.

Routing experiment. To demonstrate the flexibility of our policy based approach, we set up a 1-level DIF on top of two 0-level DIFs (Figure 5c), and we run a routing application (Procedure 1) with different link state update frequency (`f4`.) Procedure 1 shows how four frequencies can be set as a policy: `f1` and `f2` specify how often the neighbor identity and link state information are updated; `f3` specifies how frequently the link state information is published, and `f4` specifies the frequency at which the neighbors' link states are retrieved. We used two periods of 5 and 10 seconds for $\texttt{f4}^{-1}$ while we keep constant values for $\texttt{f1}^{-1}$, $\texttt{f2}^{-1}$, and $\texttt{f3}^{-1}$ to 2 seconds. The forwarding table is computed every $\texttt{T} = \texttt{f4}^{-1}$ seconds. After all the DIFs are formed and the routing tables have converged, we start a file transfer application from `IPC-A` to `IPC-D`. The file transfer starts on the flow `IPC-A`, `IPC-B`, `IPC-D`. After approximately 60 seconds, we let the intermediate process `IPC-B` be unavailable, and we observe how different routing policies update the routing tables at different speed. Not surprisingly, lower update frequency yields slower path recovery and hence lower throughput.

5.4 Application Management Case Study: Load Balancing

High performance application or content delivery networks such as the Akamai network [27] store customer states in (hierarchically) partitioned distributed storage systems. Every partition scope is limited to a subset of the private network, and appropriate storage servers are resolved using static and manually configured mapping rules. [7] Using the RINA mechanisms, this mapping can be made dynamic, and the load on such partitions can be redistributed as the utilization of storage servers change.

In the next two subsections we explain how a network monitoring application can be implemented using the RINA mechanisms for balancing both the inter and intra DIF load. We assume that a service is provided by different servers, and that each server application process registers (using its underlying IPC process) to its local IDD agent and such agent registers to the Inter-DIF Directory service.

5.4.1 Intra-DIF Load Balancing

Procedure 2 : `Intra-DIF Load Balancing`

```
1: LoadSubID = createSub('load', MemberList,freq)
2: for all Query do // to local IDD agent
3:    Load = readSub(LoadSubID)
4:    IPCName = Resolve(Query, Load, Policies)
5:    Reply(IPCName)
6: end for
```

If a client application process wants to access a service available on multiple servers within the same DIF, we let a monitoring DAF be responsible for monitoring and balancing the Intra-DIF load on the servers of the DIF. In particular, a management application can be set up as in Procedure 2.

[7] See *e.g.* the Memcached system [25].

First, the monitoring application needs to subscribe to all the members of the DIF to get updates on the load of all servers (that use communication service provided by IPC processes of this DIF) at a given frequency (Procedure 2, line 1). Such subscription is handled by the RIB daemon of each DIF member, and is added to the subscription list of processes to be updated with load information. When the IDD local agent receives a query in an attempt to resolve a given service name (line 2), the load of each server hosting the queried service is read (line 3) and compared, and the currently best IPC process name (address of an underlying IPC of the server hosing the service) is returned (line 5) based on the IDD policy such as least CPU usage (line 4.)

Intra-DIF load balancing experiment. We tested the intra-DIF load balancing by letting two IPC processes offer the same service, and enroll in the same DIF. The toy service consumes a fixed percentage of the CPU capacity for the entire duration of the experiment and we let several client application processes request the hosted service. We measure the CPU utilization of the two servers, with and without the load balancing, observing full CPU utilization on the single server when the local IDD policy redirects all requests to it, versus halved load utilization on each server when load is distributed equally across both servers (Figure 6b.)

5.4.2 Inter-DIF Load Balancing

To demonstrate the flexibility of RINA, we show how the same load balancing functionality can be performed using a different set of mechanisms. In particular, we assign to the Inter-DIF Directory (IDD) service the responsibility of balancing the load across requests for a service hosted on different DIFs.

Via their local IDD agents, services register to IDD. IDD interacts with the local agent of each DIF by subscribing to the load information of services hosted in each DIF (Procedure 3, line 1.) When an agent receives such subscription request, its RIB daemon adds the IDD to its subscribers list, and periodically publishes the load information of the requested services. In this example load balancing application, the load information is the sum of the CPU usage of all servers hosting the queried service within the DIF. Similarly to the intra-DIF load balancing application, when the IDD receives a query to resolve a certain service name (line 2), it reads such aggregate load information from its own RIB (line 3) and resolves the service name to a DIF name according to its policies (line 4.) The client IPC process can hence join the least loaded DIF to access the requested service (line 5.)

Inter-DIF load balancing experiment. We also tested the inter-DIF load balancing by letting two IPC processes hosting the same service enroll in two different DIFs. Again in this case, the same toy service consumes a fixed amount of CPU for the entire duration of the experiment and the CPU utilization of each server is compared to the CPU utilization of a single DIF / server resolution policy where the IDD redirects all the requests to a single DIF / server. This experiment demonstrates that the same result can be obtained using a different set of RINA mechanisms (Figure 6c.)

Procedure 3 : Inter-DIF Load Balancing

```
1: LoadSubID = createSub('load', agentList, freq)
2: for all Query do // to IDD
3:    Load = readSub(LoadSubID)
4:    DIFName = Resolve(Query, Load, Policies)
5:    Reply(DIFName)
6: end for
```

6. Related Work

A complete taxonomy of all the relevant architectural solutions is too large to cite fully and would require a survey; we focus on recent representative solutions along the dimensions that best clarify our contributions.

Service-Based Architectures. Online services and their ability to generate new revenue models for (service and infrastructure) providers and enterprises, together with the inability of the current Internet to cope with new requirements have recently spurred research in service-based architectures and their management. Some solutions do not rely on the current Internet architecture, *e.g.* [10, 16, 19], while others augment the TCP-IP layers, *e.g.* [26, 33]. The Serval architecture [26] for example, inserts a level of indirection (service layer between TCP and IP) to support a flow-based [8] anycast paradigm (the Internet today only supports host-based unicast), with dynamically-changing service instances: network addresses are associated with a service and are able to change over time (while Internet addresses are exposed to and cached by applications.) RINA also dynamically (re)assigns names to processes supporting the service-based anycast abstraction as in [19, 26], but (application) process names are dynamically bound to (not necessarily IP) addresses, unique within a DIF, and our RIB subscriptions naturally support anycast and multicast. We call this general form a *whatevercast* name. [9]

Extensible Control Platforms. The design and implementation of distributed systems for network control has also been an active area of research [2, 6, 11, 17, 20, 31, 34]. One example of such systems is Onix [20], a platform that pro-

[8] Flow-based means that packets of the same flow reach the same (possibly moving) service instance — set of processes — through late binding of names to addresses [36].

[9] We observe that anycast and multicast are two forms of the same construct: an anycast address is the name of a set of addresses with a rule, such that when the name is referenced, the rule is evaluated and returns *one member* of the set that satisfies the rule; a multicast address is a set where the rule returns *all members* of a group.

vides API in support of control policy implementations. As the Resource Information Base (RIB) in RINA, the Network Information Base (NIB) of Onix decouples the forwarding from the control mechanisms (avoiding their direct interaction.) RINA extends this contribution by extracting only the subset of essential interface on objects needed to perform any management operations. Moreover, unlike RINA (that provides the CDAP protocol for operations on objects), Onix does not dictate a protocol to manage network element (forwarding) states.

Another more recent solution that focuses on scalability of control applications is Kandoo [11]. By separating infrequent control operations that require network-wide state (*e.g.* routing) from frequent local control operations (*e.g.* link layer discovery), the Kandoo controller architecture splits the functionalities into a two-level design with multiple (lower-level) local controllers and a single (higher-level) centralized controller. While our RIB-API allows control of both network-wide states (with RIB subscriptions) and direct read/write operations for high frequency local control operations, the recursive nature of RINA extends the two-level controller architecture of Kandoo into a hierarchy of controllers in support of management applications that are not necessarily local, but have limited scope (*e.g.* within a single DIF.)

Network Management Protocols and Architectures. Network management has been historically split into the Open System Interconnection (OSI) paradigm, *i.e.* Common Management Information Protocol (CMIP) [15], whose goal has been to deal with distributed network management of large public telecommunication networks, and the Internet management, *i.e.* the Simple Network Management Protocol (SNMP) [3], whose management target has been limited to Internet devices. Object-based management architectures have also been proposed [23, 28], the most popular being CORBA defined by OMG [28], an attempt to unify existing paradigms supporting various management systems (*e.g.* CMIP and SNMP.)

Leveraging previous service management architecture efforts, our contribution lies in extracting their commonalities, reducing their complexity, and augmenting their architecture with support for management of more advanced services. In particular, RINA uses an object-oriented paradigm similar to the CORBA and OSI models, as opposed to SNMP that represents information only with variables, too simple to handle sophisticated management operations. The object model is however different, in the way operations on objects (CDAP as opposed to CMIP) and notifications are defined (RINA uses subscriptions).

RINA is also different in its resource specification language. Although RINA supports any type of abstract syntax, we use the Google Protocol Buffers [9] to specify data types and the managed objects, while to define data types, the OSI and SNMP models use the ASN.1 [14]. CORBA instead uses a less expressive Interface Description Language (IDL) [28].

Finally, RINA allows management processes to directly access management objects through the RIB-API as in CORBA. Through a distributed NMS implementation, RINA also supports the manager-agents paradigm, as in both the OSI and Internet models.

Declarative-Based Solutions. Relevant to our work are also solutions discussing declarative protocols [21, 22], and architectures [1, 7] for both control [8, 20] and resource management applications [7, 21, 22]. [10] In particular, the expressiveness of both wired [22] and wireless [21] policy-based protocols enables systems to adapt protocol behaviors in support of (management) applications. Most of these existing solutions tackle a specific management application, *e.g.* routing in MANET [21], or control for large scale wired networks [20]. By decoupling policies from mechanisms through the RIB, its interfaces (RIB-API) and its (CDAP) operations, RINA provides flexibility and supports policy-based dynamic service composition for a wide range of wired and wireless management applications.

7. Conclusions

The challenge of providing a simpler and more flexible network management abstraction, capable of accommodating modern service requirements and lowering the cost of network management, has fostered extensive research in service-based, network management architectures. We proposed a policy specification language within our Recursive InterNetwork Architecture (RINA), that allows dynamic network service composition, and facilitates the design and the implementation of general management applications.

We presented the design and implementation of the management mechanisms of RINA, providing a general API for network management policy programmability. We have demonstrated, through the implementation of few key common network management applications, how different policies for routing and load balancing can be specified and instantiated using different mechanisms of the architecture.

Acknowledgments

Thanks to members of the RINA team for their support and encouragement. This work was supported in part by the National Science Foundation under grant CNS-0963974.

References

[1] P2: Declarative Networking: http://p2.cs.berkeley.edu/.

[2] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking Control of the Enterprise. In *SIGCOMM '07*, pages 1–12, 2007.

[10] Routing is a management application for resource allocation.

[3] J. Case. A Simple Network Management Protocol (SNMP). ARPA RFC - 1157, May 1990. URL `ftp://ftp.rfc-editor.org/in-notes/rfc1157.txt`.

[4] J. Day. *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall, 2008.

[5] J. Day, I. Matta, and K. Mattar. "Networking is IPC": A Guiding Principle to a Better Internet. In *Proceedings of ReArch'08*, Madrid, SPAIN, December 2008.

[6] C. Dixon, A. Krishnamurthy, and T. Anderson. An End to the Middle. HotOS'09, pages 2–2, 2009.

[7] F. Esposito and I. Matta. PreDA: Predicate Routing for DTN Architectures over MANET. *GLOBECOM IEEE Global Communication conference*, 2009.

[8] F. Esposito, D. Di Paola, and I. Matta. A General Distributed Approach to Slice Embedding with Guarantees. Technical report, Boston University, TR 2012-014, 2012.

[9] Google Protocol Buffers. Developer Guide `https://developers.google.com/protocol-buffers/`, 2010.

[10] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, 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 *NSDI*, Apr. 2012.

[11] S. Hassas Yeganeh and Y. Ganjali. Kandoo: a Framework for Efficient and Scalable Offloading of Control Applications. HotSDN '12, pages 19–24, 2012. ISBN 978-1-4503-1477-0.

[12] International Telecommunication Union - OSI. Recommendation X.722: Structure of Management Information: Guidelines for the Definition of Managed Objects `http://www.itu.int/rec/T-REC-X.722-199201-I/en`, 1992.

[13] ISO/IEC 8649. Information technology - OSI - Connection-oriented protocol for the Association Control Service Element: Protocol specification, 1995. Also CCITT X.227.

[14] ISO/IEC 8824. Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation 1 (ASN.1), 1990. Also CCITT X.208.

[15] ISO/IEC 9596-1. Information Technology - OSI, Common Management Information Protocol (CMIP) - Part 1: Specification, 1991. Also CCITT X.711.

[16] V. P. J. Touch, Y-S. Wang. A Recursive Network Architecture. Technical report, USC/ISI, October 2006.

[17] J. P. John, E. Katz-Bassett, A. Krishnamurthy, T. Anderson, and A. Venkataramani. Consensus Routing: the Internet as a Distributed System. NSDI'08, pages 351–364, 2008. ISBN 111-999-5555-22-1.

[18] K. McCloghrie and M. Rose. Management Information Base for Network Management of TCP/IP-based internets. `http://www.ietf.org/rfc/rfc1156.txt`, 1990.

[19] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A Data-Oriented (and Beyond) Network Architecture. *SIGCOMM Comput. Commun. Rev.*, 37(4):181–192, Aug. 2007.

[20] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A Distributed Control Platform for Large-Scale Production Networks. OSDI, pages 1–6, 2010.

[21] C. Liu, R. Correa, X. Li, P. Basu, B. Loo, and Y. Mao. Declarative Policy-Based Adaptive MANET Routing. In *Network Protocols, 2009. ICNP 2009. 17th IEEE International Conference on*, pages 354 –363, Oct. 2009.

[22] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan. Declarative Routing: Extensible Routing with Declarative Queries. SIGCOMM, pages 289–300. ACM, 2005.

[23] Luca Deri. *A Component-based Architecture for Open, Independently Extensible Distributed Systems*. PhD thesis, University of Bern, Institute of Computer Science and Applied Mathematics, June 1997.

[24] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[25] Memcached. `http://memcached.org/`.

[26] E. Nordstrom, D. Shue, P. Gopalan, R. Kiefer, M. Arye, S. Ko, J. Rexford, and M. J. Freedman. Serval: An End-Host Stack for Service-Centric Networking. In *NSDI*, April 2012.

[27] E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai Network: a Platform for High-Performance Internet Applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, Aug. 2010.

[28] Object Management Group. Object Management Architecture Guide, Revision 2.0 OMG TC Document 92.11.1, 1992.

[29] ProtoRINA. Boston University RINA Lab. `http://csr.bu.edu/rina/protorina/`.

[30] V. Ramasubramanian, Z. J. Haas, and E. G. Sirer. SHARP: a Hybrid Adaptive Routing Protocol for Mobile Ad Hoc Networks. In *MobiHoc*, pages 303–314, 2003.

[31] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for Network Update. In *ACM SIGCOMM 2012*, pages 323–334, New York, NY, USA, 2012.

[32] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service. In *Proceedings of the ACM SIGCOMM 2012*, pages 13–24, New York, NY, USA, 2012. ACM.

[33] S. R. Srinivasan, J. W. Lee, E. Liu, M. Kester, H. Schulzrinne, V. Hilt, S. Seetharaman, and A. Khan. NetServ: dynamically deploying in-network services. In *ReArch*, pages 37–42, New York, NY, USA, 2009. ACM.

[34] A. Tootoonchian and Y. Ganjali. HyperFlow: a Distributed Control Plane for OpenFlow. In *Proceedings of the first workshop on Hot topics in software defined networks*, INM/WREN'10, pages 3–3, 2010.

[35] Y. Wang, F. Esposito, and I. Matta. Demonstrating RINA Using the GENI Testbed. In *Second GENI Research and Educational Experiment Workshop (GREE2013)*, pages 93–96, Salt Lake City, UT, USA, March 2013.

[36] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable Flow-based Networking with DIFANE. In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 351–362, New York, NY, USA, 2010.