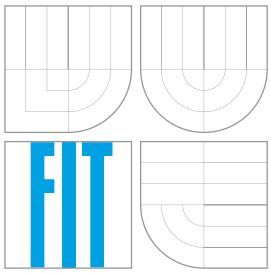
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODELOVÁNÍ NOVÝCH SÍŤOVÝCH ARCHITEKTUR V OMNET++

MODELLING NEW NETWORK ARCHITECTURES IN OMNET++

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ HYKEL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MARCEL MAREK

BRNO 2015

Abstrakt

V této práci jsou popsány principy a stav implementace vybraných nových síťových architektur. Dále je dokumentována implementace modulu Relaying and Multiplexing Task síťové architektury Recursive InterNetwork Architecture (RINA) pro simulační framework OMNeT++. Cílem práce je doplnění funkcionality již existující simulační knihovny pro zajištění plnohodnotného modelování sítí RINA.

Abstract

This thesis describes principles and state of implementation of selected new network architectures. It also documents implementation of the Relaying and Multiplexing Task module of one of the presented architectures, Recursive InterNetwork Architecture (RINA), for the OMNeT++ simulation framework. The main goal of this thesis is to extend functionality of an existing simulation library to provide a full-fledged means for modelling RINA networks.

Klíčová slova

OMNeT++, RINA, počítačové sítě, síťové architektury

Keywords

OMNeT++, RINA, computer networks, network architectures

Citace

Tomáš Hykel: Modelling New Network Architectures in OMNeT++, bakalářská práce, Brno, FIT VUT v Brně, 2015

Modelling New Network Architectures in OMNeT++

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval sám pod vedením Ing. Marcela Marka.

.....
Tomáš Hykel
May 18, 2015

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Marcelovi Markovi za podnětné rady během tvorby práce a poskytnutí možnosti podílet se na zajímavém projektu.

Contents

Chapter 1

Introduction

Today's field of computer networking and its research is heavily centered around the underlying architecture of the Internet and its protocol suite, which is known as TCP/IP for its two most prominent protocols Transmission Control Protocol (TCP) and Internet Protocol (IP). While TCP/IP remains in use for several decades and seems to work as intended, there has been a growing trend in the research community of introducing new network architectures. This thesis aims to analyze several examples of such architectures and contribute to implementation of one of them.

Network simulation is an ideal approach for examining new network architectures since it provides a quick and efficient way of setting up test scenarios and observing all aspects of their behavior. The discrete event network simulation framework OMNeT++[1] has been chosen as the target implementation platform.

1.1 Goals

The theoretical part of this thesis aims to describe some alternatives to the currently prevalent network architectures. Since the Internet is by far the largest and most important example of an internetwork, its underlying architecture shall be used as a base for comparison. This is only fitting since nearly all of the recent network architecture research is directed towards improving Internet's technology stack. Description of each architecture includes information about prototyping efforts, both in real-life platforms and in the field of network simulation.

The technical report describes design and implementation of a component of one of the presented architectures, Recursive InterNetwork Architecture (RINA), for the OMNeT++ framework.

1.2 Thesis Structure

Chapter 2 describes the shortcomings and weak parts of current Internet technology which create the need for alternative architecture research. This overview serves in the next chapter as a reference point for evaluating contributions of alternative architectures.

Chapter 3 provides a brief analysis and evaluation of several new network architectures. It also documents their prototyping efforts, both in real settings and in simulation.

[1] http://www.omnetpp.org

Chapter 4 takes a closer look at parts of RINA that are related to the implementation task of this thesis.

Chapter 5 describes implementation of RINA's Relaying and Multiplexing Task for OMNeT++.

Chapter 6 presents evaluation of the implementation in form of sample test scenarios and their outputs.

Chapter 2

Problems of The Current Internet

The Internet could be considered one of the most important technological achievements of the 20th century. It has brought a previously unimaginable degree of interconnection and information access to the whole world and its importance still keeps growing decades after its inception. Nevertheless, the very basic core of its technology was constructed over three decades ago in the era of first small experimental networks such as ARPANET and CYCLADES [13], when the the demands on internetworking capabilities were nowhere compared to now.

During the Internet's growth, whenever there was a problem that required a solution, it has been usually dealt with in a non-intrusive evolutionary fashion by applying a new principle on top of the underlying technology. In another words, problems have been mostly solved by adding a new protocol to the TCP/IP protocol stack.

The Internet's evolution can be presented on EvoArch [1], an abstract model for studying protocol stacks and their evolution. The model suggests that the Internet protocol stack resembles an hourglass (see Figure 2.1). While the top and down layers are often expanded with new protocols, the usage of the internet layer's IP protocol remains constant because it acts as a common technology for all different networks interconnected by the Internet.

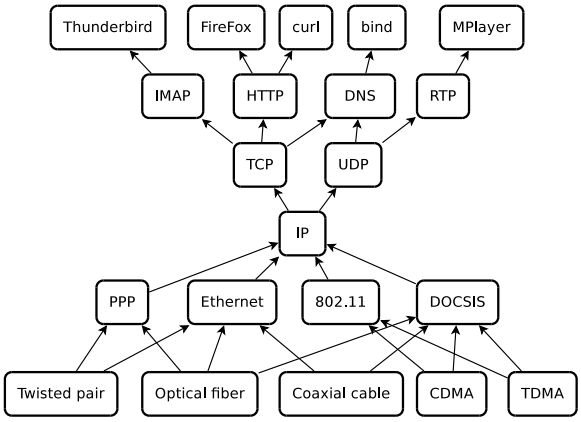


Figure 2.1: Internet's "hourglass architecture".

The evolutionary approach to improving the Internet's base technology is convenient since each paradigm shift in foundations of the Internet (i.e. replacing the "thin waist" of the hourglass) can require a long and expensive transfer of existing network configurations

to the new technology. The most notable example is the internet layer protocol IPv6 which requires explicit firmware support from active network components. The problem of IPv4 space exhaustion has been known of since the first half of 1990s [8] and the first formal IPv6 specification arose in 1998 [7], but yet, as of April 2015, four years after the top-level IPv4 pool exhaustion [2], IPv6 still represents only a miniscule fraction of the total traffic on the Internet. For example, Google IPv6 adoption statistics indicate around 6% coverage amongst the users of its services [12].

As such, some of the Internet's widely recognized problems are inherent because of the base design and it is usually difficult, if not impossible, to solve them in a non-intrusive and backward-compatible way. The following sections illustrate such problems.

2.1 Incomplete Naming Scheme

In 1982, Jerome Saltzer in his work "On the Naming and Binding of Network Destinations" [17] described the entities and the relationships that make a complete naming and addressing schema in networks. According to Saltzer, there are four elements that need to be identified: *applications*, *nodes*, *points of attachment* to the network (PoAs) and *paths*. The relationships between them are illustrated in Figure 2.2. At the time, network architectures such as CYCLADES, XNS, DECNET and OSI conformed to this scheme [19].

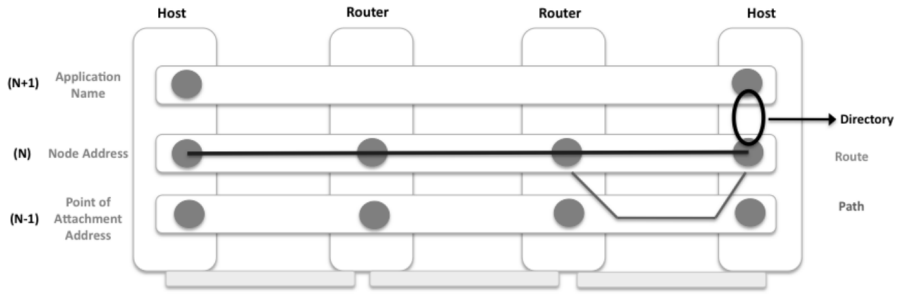


Figure 2.2: Jerome Saltzer's view of computer networking.

TCP/IP does not follow this proposal: the layer for node naming is completely missing. While TCP/IP does work with two distinct layers with their own address scopes – the link layer with physical addresses and the internet layer with IP addresses – both effectively identify the same: a host interface, i.e. a PoA address. This effectively means that the IP addressing is semantically overloaded to represent both identity and location. The need for an explicit identifier→locator mapping was eventually recognized still in the era of ARPANET and this was solved by creating a globally available file `HOSTS.TXT` containing mappings of alphabetic host names to IP addresses. Later on, this method was obsoleted by the Domain Name System (DNS). However, both approaches move the matter of node naming into the application layer, forcing applications to work with location-dependent interface PoA addresses.

The fact that the Internet forwarding is location-based instead of identity-based has a great impact on difficulty of multihoming[1] (section 2.2) and mobility (section 2.3).

[1]Multihoming refers to a computer or device connected to more than one computer network. Such computer or device generally needs a separate interface for each network.

2.2 Lack of Multihoming

Since IP addresses serve as points of attachment (i.e. they identify network interfaces) and routing is done exclusively on the internet layer, there is not any inherent mechanism for distinguishing whether multiple IP addresses identify a common node. Thus, in effect, multihoming on IP alone is not feasible.

The insufficient base for multihoming support is one of the oldest recognized problems of the Internet: it became apparent back in 1972, when Tinker Air Force Base joined ARPANET and voiced a request for redundant connections to a single node to ensure reliability [6]. In spite of this, the switch to a new protocol suite that happened 11 years later (on 1.1.1983, the "flag day") did not bring any solution to this problem.

Since then, some attemps were made to implement multihoming on top of the current architecture.

- **SCTP.** Message-oriented transport protocol Stream Control Transmission Protocol (SCTP) [18] provides a partial support: two SCTP hosts are able to provide each other with lists of fallback IP addresses that may be used in case of the primary address going offline. However, thus far, multiple reasons have been preventing SCTP from becoming a widely known and used solution; the most notable disadvantage lies in the fact that due to TCP/IP not recognizing a distinct session layer on top of its transport layer (such as in ISO/OSI stack), the transport protocol has to be explicitly specified by the application using the BSD sockets Application Programming Interface (API). Therefore, SCTP adoption would require a rewrite of network-aware applications themselves. Other SCTP adoption issues include unsatisfactory operating system support (Microsoft Windows systems require a third-party kernel driver) and weak awareness of its existence outside the networking community.

- **BGP Multihoming.** Another implementation of multihoming capabilities can be seen in Border Gateway Protocol (BGP) [16], which provides a means for load-balancing and fallback over multiple links on T1 networks. To make use of such multihoming over the Internet, a public IP address range and an Autonomous System number are required. BGP Multihoming is one of the most significant causes contributing to the growth of the global Internet routing table.

- **Multipath TCP.** The most recent TCP/IP multihoming initiative is the TCP extension Multipath TCP (MPTCP) which is currently in its experimental phase, although a large scale commercial deployment has been already made by Apple for its Siri network application in mobile operating system iOS 7 [11].

2.3 Lack of Mobility

Since nodes are identified solely by IP addresses of their interfaces (i.e. points of attachment to the network), mobility is essentialy non-existent.

There have been three distinct approaches to solving the mobility problem [20].

- **Mobility by indirection.** A fixed host or device is dedicated for keeping track of mobile devices and forwarding traffic to them. This leads to path inflation. This approach is used by technologies such as Mobile IP or Locator/Identifier Separation Protocol (LISP).

- **Global name resolution.** The endpoint identifier is resolved to its network location by looking up a logically centralized service. This approach is used by technologies such as eXpressive Internet Architecture (described in Section 3.4) or MobilityFirst (described in Section 3.3).
- **Name-based routing.** The network locator is not used at all and routing is done on names. This approach is used by technologies such as Named Data Networking (described in Section 3.2).

The second and third approaches require a clean-slate architectural design.

2.4 Lack of Security Mechanisms

The specifications of the fundamental protocols of TCP/IP stack – IP, TCP and DHCP – were originally completed at the beginning of 1980s. The Internet has since then turned into a massive world-wide internetwork connecting people of different types and agendas. Naturally, once the Internet began to be used for transferring sensitive data (especially by companies), cyber crime started to emerge as well and some attention was turned to security aspects of Internet protocol (or lack thereof).

As the IP protocol lacks any inherent verification mechanism, the internet layer is easily subjected to hijacking and spoofing. This provides opportunity for many types of exploits, most notably the Man-in-the-middle attack or IP address spoofing [4].

Since security is not enforced by the architecture in any way, it is still common even for the application layer with its plethora of protocols to be subjected to security problems. Protocols like TCP are still widely in use and remain vulnerable to attacks such as TCP sequence prediction attack [4]. Today, application protocol security is usually achieved by "wrapping" protocols into other cryptographical protocols such as Transport Layer Security (TLS) or Datagram Transport Layer Security (DTLS).

2.5 Routing Table Size Growth

Default-free zone (DFZ) is the collection of all Internet autonomous systems (ASs) that do not require a default route to forward a packet to any destination. Since they comprise the root of Internet's routing infrastructure, their database must be complete.

With the increasing number of hosts connected to the Internet, the DFZ routing table sizes grow as well (see Figure 2.3).

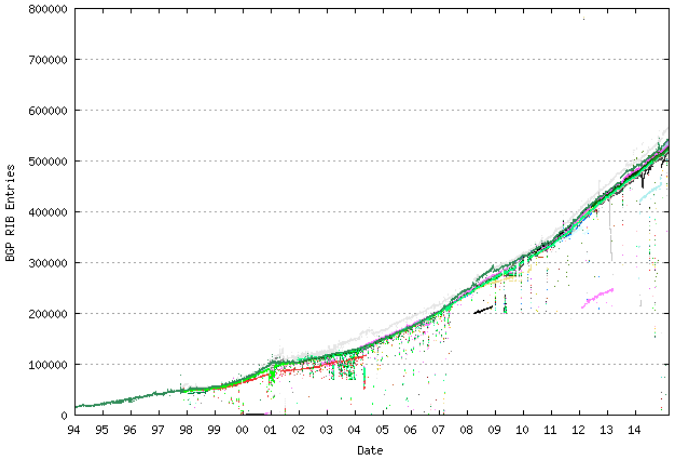


Figure 2.3: Global Internet routing table size growth [5].

While the exponential growth observed during the 1990s was later mitigated by mass deployment of Classless Inter-Domain Routing (CIDR) and BGP route aggregation, the number of items is still increasing superlinearly and the high-end router hardware needs to keep up, especially with the increasing use of BGP-based multihoming and IPv6. This can sometimes lead to scalability problems, as in August of 2014, when reaching the 512k entry limit of multiple routers caused globally observable outages [22].

As of April 2015, the Internet routing table consists of over 560k entries [5]. There are concerns about whether the technology of high-end routers will keep scaling along with Moore's law to keep route management efficient [14].

Chapter 3

Alternative Network Architectures

This chapter provides descriptions of several new network architectures.

Due to a limited scope of this thesis, the chapter describes and evaluates only a small representative subset of new architectures. The subset consists of projects funded by the Future Internet Architecture – Next Phase program [9] (Named Data Networking [23] [Section 3.2], MobilityFirst [20] [Section 3.3] and eXpressive Internet Architecture [15] [Section 3.4]) and the Recursive InterNetwork Architecture [6] (3.5).

Special attention will be given to Recursive InterNetwork Architecture as one of its components is the implementation goal of this thesis.

3.1 Design Approaches

The networking research community has exhibited many attempts of moving the field forward. The undertaken research directions are often classified into one of two groups:

- **Evolutionary design.** Backward-compatible solutions that are incrementally deployable on top of the current Internet (e.g. LISP or DiffServ).
- **Clean slate design.** Completely new standalone architectures that are not constrained by Internet technology's limitations.

Considering the scope of this thesis, the focus will be given exclusively to "clean-slate design" architectures.

3.2 Named Data Networking

Named Data Networking is one instance of a more general research direction called *Information-centric networking* (ICN) [10]. ICN explores the possibilities of moving the Internet infrastructure away from its host-centric communication paradigm towards the idea of named content.

3.2.1 Premise

During its early years, ARPANET was heavily influenced by telecommunication technology as the Public switched telephone network (PSTN) was the only example a large-scale

network. Due to this, technology of the Internet has been built on the paradigm of end-to-end communication based on network addresses. However, while this base paradigm has remained constant over the decades, the way we use the Internet has considerably changed: the Internet is now used primarily as a content distribution network. Since the mechanism of data retrieval over the Internet is based on creating end-to-end communication channels and transferring data through them, the content itself is transparent to the network and this generates an enormous amount of data redundancy.

Named Data Networking proposes a solution more fitting for today's needs: instead of working with the source/destination addresses, the "thin waist" of the Internet should be based on working with names of data chunks (as illustrated by Figure 3.1).

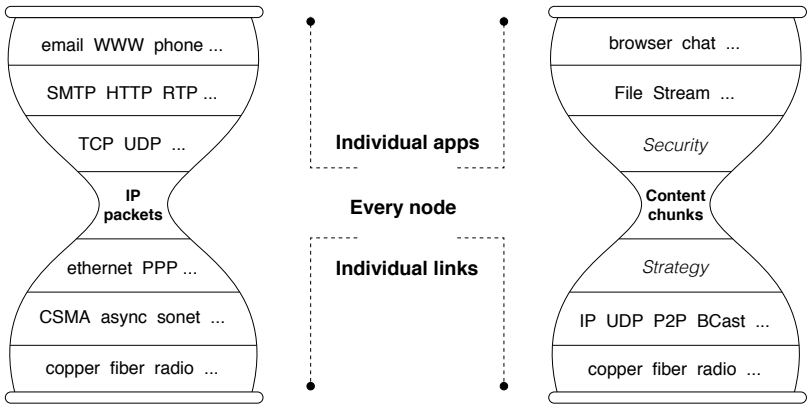


Figure 3.1: "Thin waists" of the current Internet and NDN.

3.2.2 Concepts

The Building Blocks

The NDN architecture specifies:

- two types of packets: an *interest packet* containing the name of desired data and a *data packet* containing the requested data (shown in Figure 3.2),

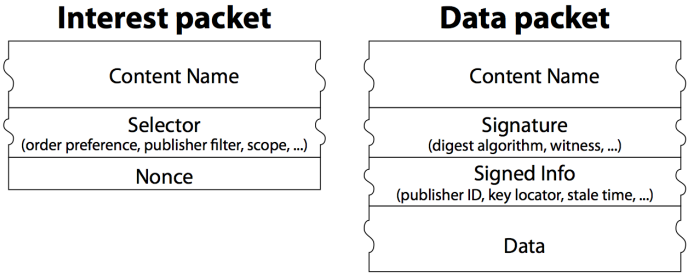


Figure 3.2: NDN packet types.

- two types of hosts: a *consumer* (data requester) and a *producer* (data provider), and
- a *router* maintaining three fundamental data structures:

- *Forwarding Information Base* (FIB) (forwarding table)
- *Pending Interest Table* (PIT) (data request management)
- *Content Store* (data cache)

The Communication Model

Communication in NDN is driven by the data receiver, i.e. the consumer. The steps are:

1. The consumer sends out an interest packet containing the name of the desired data.
2. When a router receives the interest packet, it first consults its Content Store for requested data.
   - If the data requested by the interest packet are present, they are returned in the direction of the requesting interface.
   - Otherwise, it'll look up the PIT.
     - If there's an entry present for the named data request, the entry is updated by adding the originating interface into the list of requesting interfaces, thus aggregating the new request together with an existing one.
     - Otherwise, a new entry is inserted, a FIB lookup is made and the interest packet is forwarded to interface(s) returned by the FIB.
3. A data packet is returned to the router by either the producer or another router with cached data. The router finds a matching PIT entry and forwards the data to all interfaces listed in the PIT entry. The PIT entry is then removed and data are cached into the Content Store.

Naming

NDN assumes data chunk names to be hierarchically structured. Consumers must be able to deterministically construct the name for a desired piece of data without having previously seen the name or data. This can be achieved by a deterministic algorithm allowing both consumer and producer to construct the same name based on data available to both.

The management of such namespace is not defined by the architecture itself and should be a subject of further research.

3.2.3 Current State of Implementation

NDN's implementation efforts are open-source and available as a package called `NDN Platform`[1]. The package contains a C++ library (`ndn-cxx`), the NDN Forwarding Daemon (`NFD`), client libraries for C++, Python, Java and JavaScript, NLSR routing protocol, NDN repository and additional networking tools (a ping-like application, a traffic generator and a traffic capture tool).

`ndnSIM`[2], based on `ns-3`[3], is a network simulator using `ndn-cxx` and `NFD` as the architecture backend. `ndnSIM` extends `ns-3` with a new network-layer protocol model which can be used atop of any available link-layer protocol, thus providing a flexible solution for simulating various development scenarios.

[1] http://named-data.net/codebase/platform/
[2] http://ndnsim.net/2.0/
[3] https://www.nsnam.org/

3.3 MobilityFirst

3.3.1 Premise

The current Internet is designed for interconnecting fixed endpoints and fails to address dramatically increasing demands of mobile devices and services. MobilityFirst, as its name would suggest, aims to provide a means for better mobility, while also introducing intristic security properties and faciliating services.

3.3.2 Concepts

MobilityFirst is based on three basic principles: separation of locator and identifier (i.e. node name and PoA address), intristic security and global name resolution.

Locator/Identifier Separation

MobilityFirst's "thin waist" consists of location-independent names and a global name service for mapping them to addresses. A name is a *globally unique identifier* (GUID) that can be used to identify a variety of *principals* such as an interface, a node, a service, an end-user or content. An example of a principal is a *network address* (NA), a network identifier resembling Internet's autonomous system.

Intristic Security

GUIDs are self-certifying, so any principal can authenticate another principal without relying on an external authority. This is achieved through bilateral challenge-response mechanism.

e.g. Principal `X` wants to verify authenticity of principal `Y` before establishing a connection to him.

1. `X` sends a random nonce `n` to `Y`
2. `Y` responds with $\{pubkey, privkey(nonce)\}$
3. if $hash(pubkey) = Y$ and $pubkey(privkey(nonce)) =$ `n`, `Y` is authenticated

Name Resolution

MobilityFirst defines a naming service for dynamic mapping of GUIDs to network addresses with real-time response latencies. Unlike today's DNS with its reliance on a single root authority (ICANN), the naming service is decentralized.

In addition to this, a principal can also be assigned an optional human-readable name which is bound to its public key by a name certificate. In this case, the certificate has to be obtained from a trusted certification authority.

The system encompassing both the name resolution service and the name certification service is called the *Global Name System* (GNS).

The Communication Model

1. To contact a GUID, the sending endpoint queries the GNS to obtain an NA corresponding to a GUID (much like it queries DNS to obtain an IP address for a domain name).

2. The sending endpoint then begins sending data, using the tuple `[GUID, NA]` (which is a routable destination identifier) in packet headers.

Senders can also send a packet addressed just to a GUID, thereby implicitly delegating to the first-hop router the task of querying the name service for an NA.

3.3.3 Current State of Implementation

The MobilityFirst prototype is available by request to project leaders and consists of following components:

- `msocket`, an endpoint socket library extending the BSD sockets API.
- `Auspice`, a GNS implementation.
- Two prototypes of the forwarding plane: one based on the Click modular router[4], other based on OpenFlow.

There is no tool available for simulation.

3.4 eXpressive Internet Architecture

3.4.1 Premise

As presented in the previous sections, some of the future architecture research is centered around the idea of replacing Internet's "thin waist" of end-to-end communication with a different principal or a set of principals (e.g. NDN and its named content). eXpressive Internet Architecture (XIA) takes this approach one step further and proposes a novel principle: the "thin waist" should provide support for multiple principals and the ability to evolve by accomodating new principals over time.

3.4.2 Concepts

XIA is built around three basic principles: evolvable thin waist (achieved by configurable principal types), intristic security and flexible addressing mechanism (achieved by DAG addressing).

Principal Types

An *XIA principal* is specified by the semantics of communicaton between principals of the same type, the processing that is required to forward traffic with addresses of its type and a unique *XIA identifier* (XID). The initial XIA architecture defines four basic XIA principal types:

- **Host XID (HID).** HIDs support unicast host-based communication similar to IP where the host identifier is a hash of the host's public key. HIDs define who you communicate with.
- **Service XID (SID).** SIDs support communication with (typically replicated) services and realize anycast forwarding scheme. SIDs define what entities do.

[4] http://www.read.cs.ucla.edu/click/click

- **Content XID (CID).** CIDs allow hosts to retrieve content from "anywhere" in the network, e.g., content owners, CDNs, caches, etc. CIDs are defined as the hash of the content, so the client can verify the correctness of the received content. CIDs define what it is.
- **Network XID (NID).** NIDs specify a network, i.e., an autonomous domain, and they are primarily used for scoping. They allow an entity to verify that it is communicating with the intended network.

Apart from the above, other basic types have been introduced and experimented with, e.g. 4IDs replicating IPv4 addresses.

Intristic Security

Security properties are included in principal type definitions and entity validation is achieved through use of self-certifying identifiers in a manner similar to MobilityFirst (see Section 3.3.2).

DAG Addressing

Addressing in XIA is realized by using Directed Acyclic Graphs (DAGs). In their simplest form, address DAGs may be used only for specifying packet's destination ID as in traditional network architectures (3.3a). However, in addition to that, they can also contain scoping information (e.g. target network + a service located in the network [3.3b]) or fallback alternatives (e.g. Network XID to be used by forwarding when given SID is not recognized [3.3c]).

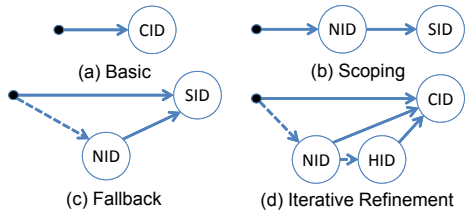


Figure 3.3: DAG-based addressing in XIA.

3.4.3 Current State of Implementation

XIA's prototypes are open-source and publicly hosted on GitHub. One is a Click-based prototype which includes the XIA protocol stack, a routing daemon, a nameserver and an API. Another is a native Linux network stack implementation with attempts to port different architectures to XIA.

There is no tool available for simulation.

3.5 Recursive InterNetwork Architecture

Recursive InterNetwork Architecture (RINA) is an architecture based on a set principles described by John Day in his book *Patterns in Network Architecture* [6].

3.5.1 Premise

RINA introduces a new perspective on computer networking:

> Computer networking is a recursively scalable set of distributed applications specialized to do inter-process communication.

3.5.2 Concepts

RINA specifications define a rich set of new concepts based mostly on theory of distributed computing. They are described in the following sections.

Distributed Application Facility

Distributed Application Facility (DAF) is a collection of two or more cooperating application processes in one or more computing systems, which exchange information using *inter-process communication* (IPC) and maintain shared state.

Distributed IPC Facility

Distributed IPC Facility (DIF) is a collection of two or more application processes cooperating to provide IPC. A DIF's application is called an IPC process and DIF is a DAF that does IPC. The DIF provides IPC services to application processes via a set of API primitives that are used to initiate flow and exchange data with the application's peer.

Since DIFs are conceptually DAFs as well, their application processes can also exchange information using other DIFs. This yields a recursive structure.

A DIF is essentially RINA's equivalent of an abstraction layer. However, unlike the traditional network architectures such as the seven-layer ISO/OSI, RINA has only one layer which "vertically repeats". This also requires a different kind of notation: instead of using absolute layer identifiers such as *Layer 2* or *Layer 7*, RINA's DIFs are referred to in a relative manner in relation to a specific level, e.g. *(N)-DIF*, *(N+1)-DIF* or *(N-2)-DIF*.

A computing system may be a member of $< 0, n >$ DIFs and has a separate IPC process for each DIF. Each (N)-DIF handles data coming from (N+1)-DIFs in the same way as from an application.

A bare minimum for a computer internetwork consists of three levels of DIFs and three types of devices: a *host*, an *interior router* and a *border router*. The internetwork is shown in Figure 3.4.

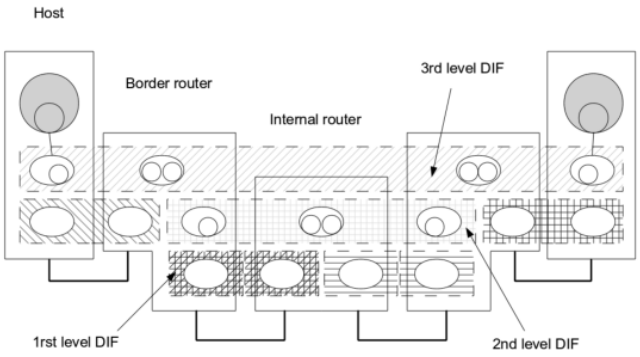


Figure 3.4: An example of a RINA internetwork with 3 levels of DIFs.

As the architecture is recursively scalable, this model can be expanded further: for example, other DIFs can be added on top of the stack in hosts for creating another scope of communication (e.g. for VPN-like facilities).

DIF operates in its own scope isolated from other DIFs of the same level. Therefore, DIF maintains its own distinct namespace and configuration (such as policies related to security and data transfer). When a computing system wants to communicate with another system inside a foreign DIF, it needs to go through a process of enrollment to the DIF first.

IPC Process

Each IPC process executes routing, transport, security/authentication and management functions. The components of an IPC process responsible for providing these functions can be categorized under three decoupled parts operating at separate timescales: *IPC transfer*, *IPC control* and *IPC management* (see Figure 3.5). The behaviour of each part can be configured via policies.

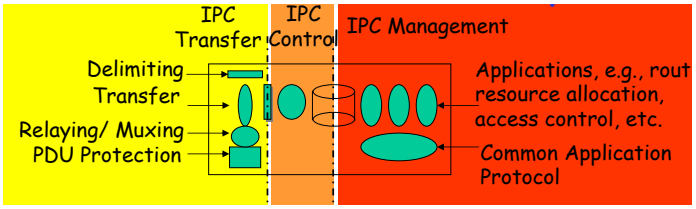


Figure 3.5: Parts of RINA's IPC process.

- IPC transfer consists of following modules:
  - ***Delimiting.*** Marks boundaries on incoming (N+1)-SDUs.
  - ***Error and Flow Control Protocol* (EFCP).** A Delta-T [21] based protocol that handles individual data flows. In general, EFCP instances exchange data with each other via PDUs.
  - ***Relaying and Multiplexing Task* (RMT).** A stateless function that 1) retrieves PDUs from EFCP instances and management tasks and multiplexes them

onto a common (N-1)-flow, and 2) takes incoming PDUs and relays them within current IPC or passes them to outgoing port(s).

- IPC control is an optional mechanism handling flow control (for example, TCP-like flow control could be implemented here via appropriate policies). Its functionality is encompassed in the EFCP module.
- IPC management handles management tasks such as routing, resource allocation or access control. It consists of following modules:
  - ***Resource Allocator* (RA).** Monitors the resource allocation and performance of the IPC Process and makes adjustments to its operation to keep it within the specified operational range.
  - ***Resource Information Base daemon* (RIB).** The heart of DIF management. Receives/sends management messages and notifies other submodules about management changes.

Addressing

Since each DAF operates within its own address scope, the architecture needs to provide a means of resolving (N)-application names to addresses of (N-1)-IPC processes. Such mappings are stored in IPC process's Directory. Directories are managed by a decentralized distributed *Name Space Manager* (NSM) embedded in each DIF.

The NSM maintanence is one of the tasks of IPC management; each IPC process keeps track of local registered applications and some of them are specialized to maintain aggregated repository of non-local mapping to serve as forwarders (such processes are called *Repository IPC Processes*). This is illustrated in Figure 3.6.

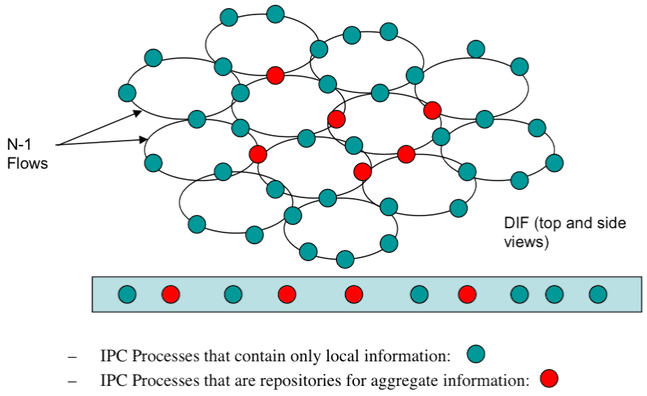


Figure 3.6: Distributed mapping of applications to addresses in a RINA network.

The Communication Model

A connection between two applications in RINA needs to go through the initial process of flow allocation.

When an application named `App1` wants to create a flow with other application named `App2` reachable via a common DIF, the Allocate procedure proceeds as follows:

1. `App1` requests an IPC connection to `App2` with desired QoS requirements. This request is handled by Flow Allocator of the underlying IPC process with address `IPC 1`.
2. The Flow Allocator validates the request. If the request is well formed and the IPC process has enough resources to honor it, it is accepted and an EFCP instance is created. Otherwise, an error is returned.
3. The Flow Allocator asks Directory for address of the IPC Process to which `App2` is mapped. In this case, it is `IPC 2`.
4. The Flow Allocator asks `IPC 1`'s Resource Allocator to find a suitable (N-1)-flow to map the new flow to it. If there is not any, RA requests an IPC connection to `IPC 2`.
5. The Flow Allocator instructs RIB to send out an `M_CREATE` request to `IPC 2`.
6. Upon the request receival, the `IPC 2`'s Flow Allocator delivers the Allocate request to App2. If App2 submits a positive response, the Flow Allocator creates an EFCP instance.
7. `IPC 2`'s Flow Allocator instructs RIB to send out `M_CREATE_RESPONSE` request back to `IPC 1`.
8. `IPC 1`'s RIB receives the `M_CREATE_RESPONSE` request. If it is positive, the Allocate() procedure is complete.
9. `App1` and `App2` can now use the IPC API to send and receive data SDUs to/from each other. When the communication is over, both of them can invoke the Deallocate call to release allocated resources.

Policy Separation

RINA heavily relies on the design approach of separating mechanism and policy: parts related to authorization of operations and allocation of resources remain constant, while the decisions regarding how to use them are left to policies. In effect, there's only one application protocol (CDAP) and one error and flow control protocol (EFCP). A clear example of the policy separation is EFCP, which is a single mechanism configurable for both reliable and unrealiable data transfer, as opposed to TCP and UDP, which are two distinct mechanisms.

Because of this, RINA can serve as a platform for evaluating multiple approaches to a given problem just by replacing policies. An example can be observed in RINA's approach to routing, which is a policy by itself, providing a platform for implementing well-known protocols (such as those based on distance vector or link-state algorithms) as well as experimental new paradigms (such as hierarchical and topological routing).

3.5.3 Current State of Implementation

`ProtoRINA` is a Boston University's reference prototype written in Java, publicly available for download from the project's pages. `IRATI Stack` is an open-source attempt to port RINA into the Linux kernel network stack.

`RINASim` is an open-source OMNeT++ implementation of RINA developed by FIT BUT under the PRISTINE[5] project. It is publicly hosted on GitHub.

[5] http://ict-pristine.eu

3.6 Evaluation

In the previous sections, I have described 4 new network architectures. This section puts them into context of problems mentioned in Chapter 2 and points out their main strengths and weaknesses.

3.6.1 Named Data Networking

The most significant advantage of NDN is its native support for caching all sorts of data inside the network itself. While this should be beneficial mostly for static data such as web pages and images, dynamic content can take advantage of this as well in case of multicasting or packet retransmission on packet loss.

With its named data paradigm, NDN renders the problems of node naming, mobility and multihoming irrelevant, because data names remain the same no matter the locaton. Same case with security as each data chunk is required to be cryptographically signed by architecture itself.

The router table size growth presents a challenge as the NDN namespace is unbounded and addressing of named data implies that much more identifiers need to be used for global routing (by some estimations, the number of items it the global routing table might end up even 4 orders of magnitude higher [3]).

3.6.2 MobilityFirst

MobilityFirst solves the problem of locator/identifier conflation by introducing routable GUIDs and a service for mapping them to network addresses. This in effect eliminates the problem with multihoming and mobility as the traffic may be easily rerouted in case one of the interfaces of a host becomes unavailable. The security is enforced by the architecture due to its concept of self-certifying identifiers and bilateral challenge-response mechanism.

The concept of routing on NAs and GUIDs appears to mimic the currently used routing on IP network prefixes and host identifiers. However, MobilityFirst attempts to improve this by researching internetwork routing design based on small number of levels of hierarchy.

3.6.3 eXpressive Internet Architecture

XIA's answer to the locator/identifier problem (and, in turn, the multihoming/mobility problem) are the HID principals used to identify a node by a hash of its public key. Furthermore, multiple approaches to ensuring mobility can be implemented thanks to the flexibility of DAG addressing. Security is achieved by self-certifying principals.

The the number of principal types might rise over time and this presents a challenge in regards to routing table size growth. The XIA researchers are experimenting with a forwarding table scheme that should be capable of scaling to billions per entries while saturating 80 gigabytes per second [15].

The novel concept of evolvable "thin waist" supporting multiple paradigms at a time requires a control plane capable of handling diversity and incremental deployment. This will be the main aim of further XIA research efforts.

3.6.4 Recursive InterNetwork Architecture

In RINA, the problem of mobility/multihoming is solved inherently by providing a complete naming and addressing schema and introducing a distributed service for mapping

application names to IPC addresses. RINA's scaling by recursion promises to deliver much greater global routing scalability than the current Internet. Furthermore, the concept of autonomous DIFs imply security by isolation.

In comparison to the other presented architectures, RINA presents the most radical paradigm shift by disregarding a great deal of common computer networking knowledge and building a new set of principles from the ground up. While this yields the most complete and architecturally clean solution (all of the problems presented in Chapter 2 are solved inherently without aiming for them from the start), it also means that the biggest hurdle facing adoption of RINA might lie in unwillingness of networking community to accept its way.

Chapter 4

Forwarding In RINA

This chapter takes a closer look at components of Recursive InterNetwork Architecture that are related to the implementation target of this thesis. This includes a conceptual description of the forwarding and routing principles in RINA and what role RMT plays in it.

4.1 Distinction of Forwarding And Routing

Each IPC process has to solve the forwarding problem: given a set of EFCP PDUs and a number of (N-1)-flows leading to various destinations, to which flow(s) should each PDU be forwarded? In RINA, the decision is handled by the Relaying and Multiplexing Task and its forwarding policy. The action may consist of looking up the PDU's destination in a forwarding table (resembling the forwarding mechanism in traditional TCP/IP routers), but it is not a requirement; other experimental forwarding paradigms, such as forwarding based on topological addressing, may not require a forwarding table at all.

Generating information necessary to do forwarding is one of the tasks of IPC process's Resource Allocator, namely its subcomponent called PDU Forwarding Generator. For this purpose, Resource Allocator generally uses pieces of information provided by other sources, most notably the Routing Policy.

The Routing Policy exchanges information with other IPC Processes in the DIF in order to generate a next-hop table for each PDU (usually based on the destination address and the id of the QoS class the PDU belongs to). The next-hop table is then converted into a PDU Forwarding Table with input from the Resource Allocator's PDU Forwarding Generator, by selecting an (N-1)-flow for each "next-hop". The Routing Policy may resemble distance vector and link-state routing protocols used in today's Internet, but the current research is also aimed at other paradigms such as topological/hierarchical routing, greedy routing or MANET-like routing.

4.2 Relaying and Multiplexing Task

4.2.1 Formal description

RMT has, as its name suggests, two main reponsibilities: relaying and multiplexing of PDUs. The goal of multiplexing is to pass outgoing PDUs (from EFCP instances and management tasks) to the appropriate (N-1)-flows and pass incoming PDUs (from (N-1)-flows) to EFCP instances and management tasks. Relaying handles incoming PDUs from

(N-1)-ports[1] that are not directed to its IPC process and forwards them to other (N-1)-ports using information provided by its forwarding policy. A conceptual model of RMT is presented in Figure 4.1.

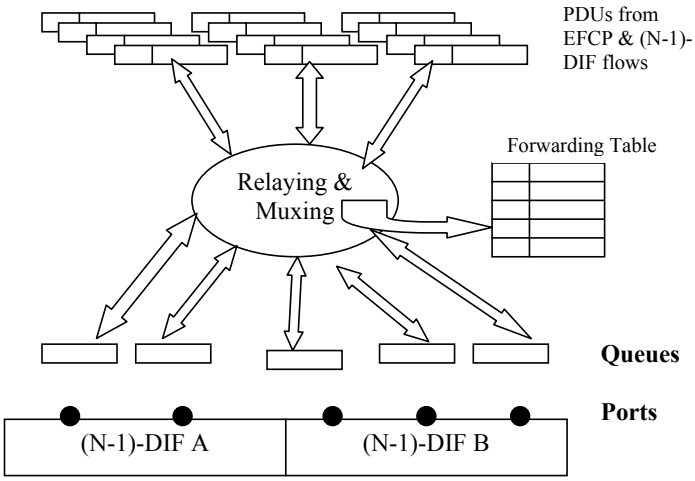


Figure 4.1: Diagram of Relaying and Multiplexing Task.

RMT instances in hosts and bottom layers of routers usually perform only the multiplexing task (Figure 4.2a), while RMTs in top layers of interior routers (Figure 4.2b) and border routers (Figure 4.2c) do both multiplexing and relaying.

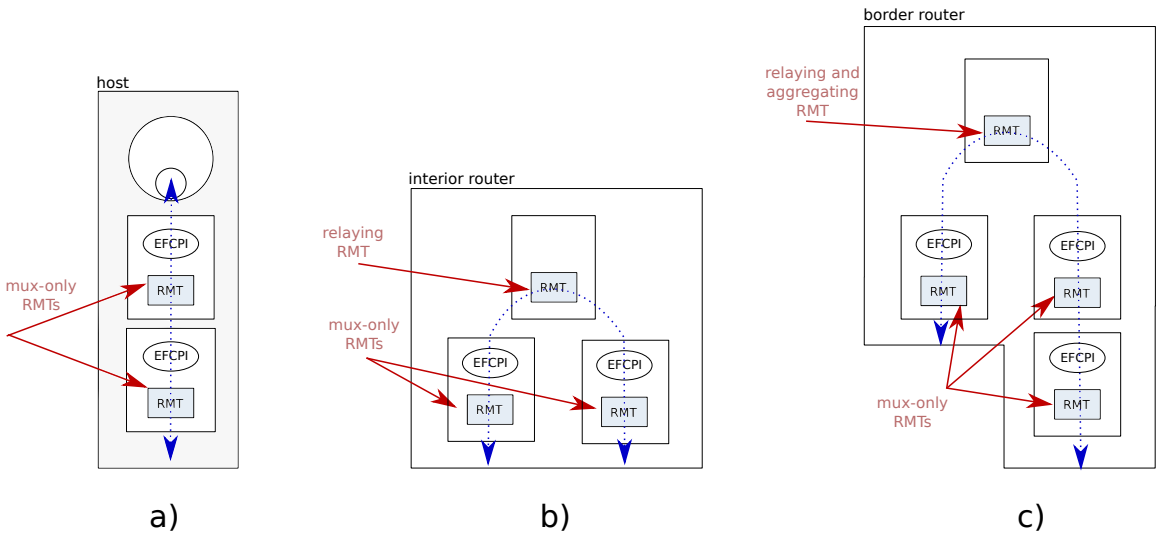


Figure 4.2: A simplified view of data flow direction in different types of devices.

Each (N-1)-port handled by RMT has its own set of input and output buffers. The number of buffers, their monitoring, their scheduling discipline and classification of traffic into distinct buffers are all matter of policies.

[1] A handle for an (N-1)-flow, not unlike the traditional BSD socket.

RMT is a straightforward high-speed component. As such, most of its management (state configuration, forwarding policy input, buffer allocation, data rate regulation) is handled by the Resource Allocator which makes the decisions based on observed IPC process performance.

4.2.2 Policies

Even though the Relaying and Multiplexing Task serves as a low-overhead component similar to the traditional view of router data plane, several policies are defined for modifying its behavior.

- **Scheduling policy.** A scheduling algorithm (also commonly known as "network scheduler algorithm" or "queueing discipline") that determines the order in which input and output buffers are serviced. This policy should be invoked each time a PDU needs to be taken from a queue for processing and works for both input and output directions. Examples of possible algorithms could be FIFO, LIFO or fair queueing.

- **Monitoring policy.** A state-keeping queue monitoring algorithm that is invoked each time a PDU enters or leaves a queue. This policy should compute variables to be used in decision process of other policies. Examples of such variables could be average queue length or queue idle time, which are often used by congestion prevention mechanisms.

- **MaxQ policy.** An algorithm that is invoked each time the number of PDUs waiting in a queue exceeds the queue's threshold. This policy is used mostly for implementing congestion avoidance mechanisms (e.g. by dropping or marking the last PDU in a queue).

- **Forwarding policy.** An algorithm used for deciding where to forward a PDU. The policy is given the PDU's *Protocol Control Information* (PCI) and in turn returns a set of (N-1)-ports to which the PDU has to be sent. This provides enough granularity to implement multiple communication schemes apart from unicast (such as multicast or load-balancing), because the decision is left to the policy. E.g. a simple forwarding policy would return a single (N-1)-port based on PDU's destination address and QoS-id, whereas in case of a load-spreading policy and multiple (N-1)-ports leading to the same destination, the policy could split traffic by PDUs' flow-ids and always return a single (N-1)-port from the set.

Chapter 5

Implementation of Relaying and Multiplexing Task

This chapter documents the implementation of RINA's Relaying and Multiplexing Task for the RINASim library.

5.1 OMNeT++

OMNeT++ is an open-source discrete event simulation framework used primarily in the field of network simulation. In this context, the adjective "network" refers to the more general meaning of the word, which means that apart from modelling TCP/IP networks (especially in conjecture with the INET library), it also provides a means for modelling other networked systems such as on-chip networks or queuing networks. As we are implementing a clean-slate architecture from the ground up, this is an ideal approach.

OMNeT++ provides a component architecture for models. Components (simple modules) are programmed in C++, then assembled into larger components (compound models) and interconnected using the high-level language NED. All events are message-driven. In theory, there are no scalability limits for networks modelled in NED and the only constraint is given by computing platform processing power.

5.2 RINASim

RINASim, developed by a networking research group at Faculty of Information Technology of Brno University of Technology, is an open-source OMNeT++ library developed for project PRISTINE. The purpose of the library is to provide a framework for modelling RINA networks and observing their behavior. In the current stage of its development, RINASim is used primarily by other PRISTINE researchers to experiment with the architecture and efficiently evaluate their working theories.

The library is open-sourced with the MIT licence and publicly hosted on GitHub.

5.3 Implementation Design

In RINASim, all functionality of RMT including a policy architecture is encompassed in a single compound module named `relayAndMux` which is present in every IPC process. The module serves for (de)multiplexing, relaying and aggregating PDUs of data flows traversing

the IPC processes. The RINASim's compound module for an IPC process can be seen in Figure 5.1.

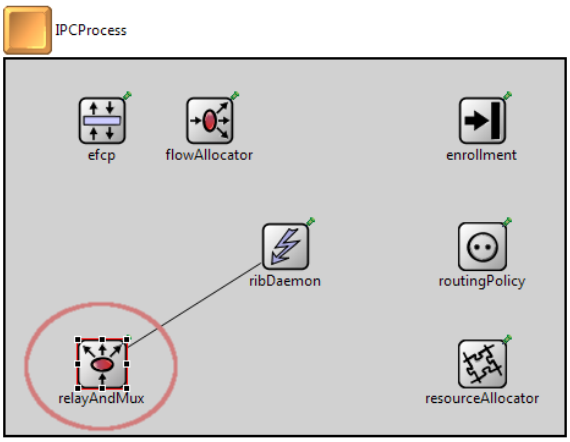


Figure 5.1: RINASim's IPC process

5.3.1 Module Structure

`relayAndMux` (an instance of compound module type `RelayAndMux`) consists of multiple simple modules of various types, some of which are instantiated only dynamically at runtime. Types of modules start with a capital letter while instances start with a small letter. Figure 5.2 presents state of the `relayAndMux` module when working with two allocated (N-1)-flows and their queues.

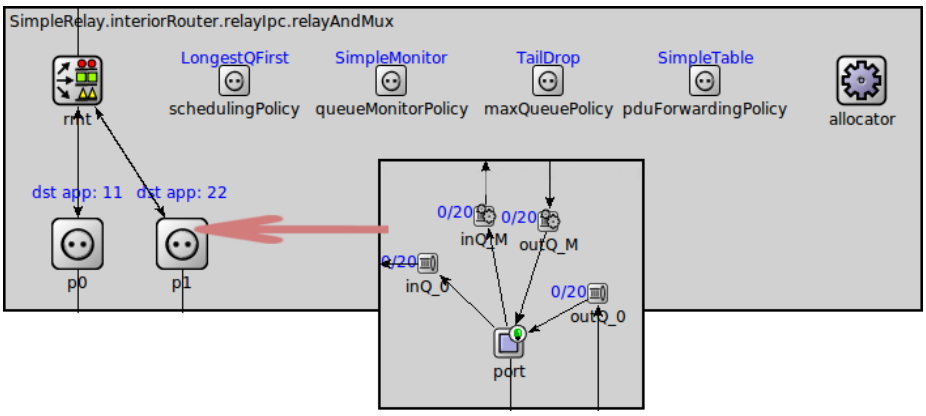


Figure 5.2: Contents of `relayAndMux` and one of its ports.

Static modules

- `rmt`, the central logic of Relaying And Multiplexing task that decides what should be done with messages passing through the module.
- `allocator`, a control unit providing an API for adding and deleting instances of dynamic modules (RMTQueue, RMTPort).

- `schedulingPolicy`, the Scheduling policy of an RMT instance.
- `monitorPolicy`, the Monitoring policy of an RMT instance.
- `maxQueuePolicy`, the MaxQ policy of an RMT instance.

Types of dynamic modules

- `RMTPort`, a representation of one endpoint of an (N-1)-flow.
- `RMTQueue`, a representation of either input or output queue. The number of RMTQueues per RMTPort is determined by Resource Allocator policies.
- `RMTPortWrapper`, a compound module encapsulating an (N-1)-port and its queues.

A class diagram showing implementation of both static and dynamic modules can be seen in Appendix B.

5.3.2 Module Parameters

The RMT module contains several user-configurable parameters that can be used to alter its behavior. These are presented in Table 5.1.

data type	name	function
`string`	`schedPolicyName`	name of the desired Scheduling policy
`string`	`qMonitorPolicyName`	name of the desired Monitoring policy
`string`	`maxQPolicyName`	name of the desired MaxQ policy
`string`	`ForwardingPolicyName`	name of the desired PDU Forwarding policy
`int`	`defaultMaxQLength`	default maximum length of instantiated queues
`int`	`defaultThreshQLength`	default threshold length of instantiated queues
`bool`	`pduTracing`	a switch for turning on PDU tracefile generation

Table 5.1: RMT module parameters.

5.3.3 Module Workflow

The core function of RMT is driven by two algorithms: the *decision algorithm* and the *dispatcher algorithm*.

The decision algorithm decides what to do with the incoming PDU and executes appropriate policies. A state diagram representation can be seen in Figure 5.3. It is implemented by `RMT::handleMessage()` and methods that are subsequently called from it (`RMT::portToEFCPI()`, `RMT::portToPort()` etc.).

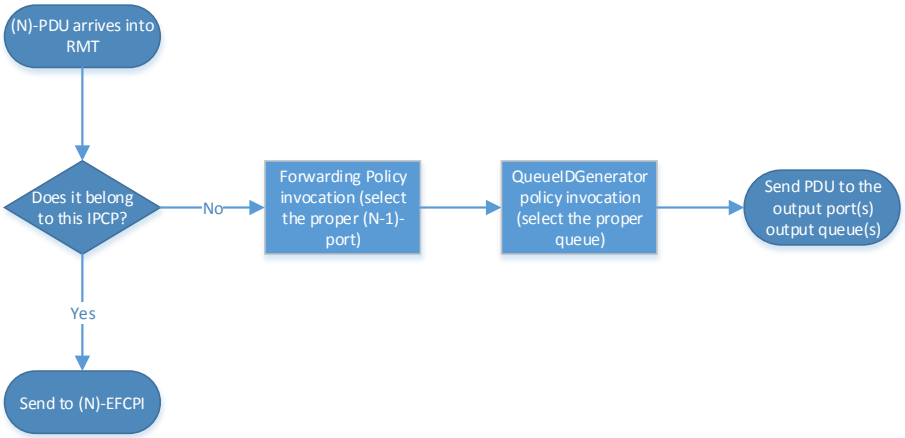


Figure 5.3: The RMT decision algorithm.

The dispatcher algorithm ensures continuous invocation of the Scheduling policy whenever there are PDUs waiting in queues. The decision of what queue should be processed next is left to the policy. A petri net representation of an algorithm used for both input and output direction can be seen in Figure 5.4. The algorithm is implemented by methods `onQueueArrival()`, `postQueueDeparture()`, `writeToPort()` and `readToPort()`.

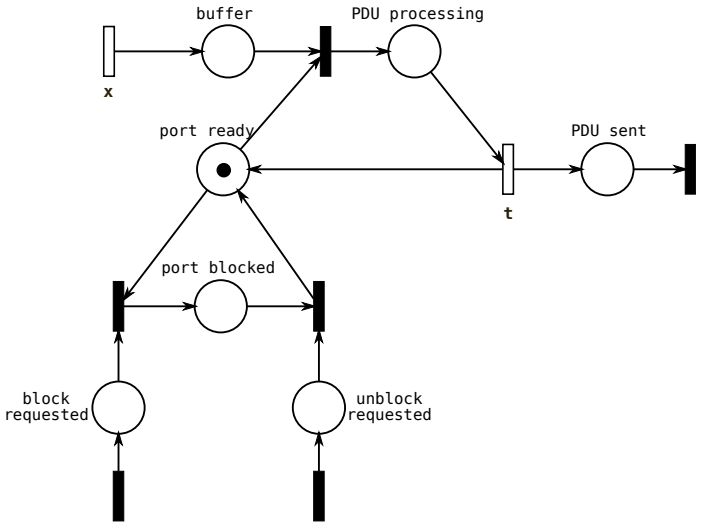


Figure 5.4: A petri net representation of the RMT dispatcher algorithm.

PDUs arrive at rate `x` into port's queues. Each time the port is ready to read/write, the scheduling policy is invoked to choose which queue should be processed, then a PDU is transmitted for time interval `t`. For output, `t` is determined by the PDU's size and characteristics of the medium; for input, it is configurable. A port block/unblock may be requested by RMT (for input) or by (N-1)-EFCPI (for output); in such case, the first ever request has to be for blocking and there cannot be multiple consecutive calls of only block requests or only unblock requests.

5.3.4 Module Management

RMT's purpose in an IPC process is fairly straightforward: providing a stateless function for relaying PDUs to their predetermined destinations and multiplexing PDUs of multiple data flows onto a common predetermined medium. The entire management of the RMT is decoupled and exercised by the Resource Allocator.

As Resource Allocator lacked any concrete specifications at the time of implementation, I have designed a set of management mechanisms, some of which are customizable by policies.

Initial RMT Mode Setup

When a RMT instance is located inside a bottommost IPC process that does not work with any further (N-1)-IPC processes, the RMT is switched to an `onWire` mode that functions over a single serializing medium instead of an IPC connection.

If a RMT instance is located inside an IPC process that has the `relay` configuration parameter configured as `true`, the RMT's Relaying Task is enabled by calling `enableRelay()`. The default value of the `relay` parameter is `true` in top layers of routers, `false` everywhere else.

Forwarding Table Management

The content of the PDU forwarding table is generated by the PDU Forwarding Generator policy module which generally accepts input from other sources such as the Routing policy module.

Queue Allocation

PDUs traversing an (N-1)-port may be momentarily buffered in input or output queues; the number of input and output queues per (N-1)-port and assignment of traffic classes to queues (e.g. all-in-one or fair queuing) is determined by two Resource Allocator policies.

- **QueueAlloc.** The (N-1)-port queue allocation strategy. The interface contains a set of event hook methods (`onPolicyInit`, `onNM1PortInit`, `onNFlowAlloc`, `onNFlowDealloc`) that allow the user to specify how many queues should be allocated or deallocated in response to which events.
- **QueueIDGen.** A companion classification policy to QueueAlloc which generates a queue ID for given PDU. This policy is used by RMT when it needs to determine which of the port's queue should a PDU be placed in.

(N-1)-port Control

Since Resource Allocator manages (N-1)-flows leading to other IPC processes, it also provides (N-1)-ports (or handles) for RMT.

In some scenarios, it may be required for an (N-1)-port to cease/slow down sending or providing more data because of congestion. Resource Allocator can momentarily disable or slow down data rate on distinct ports if this is required by EFCP instances.

5.3.5 Statistics Collection

OMNeT++ modules provide a means for declaring scalar or vector NED variables used for statistics collection. Processing of such statistical data (e.g. generating summaries and graphs) is decoupled from the act of data collection itself, so it is up to the user to pick out which data he wants to work with.

Since the Relaying and Multiplexing Task is the shared point of data flow traversal in the IPC process, it is well-suited for monitoring data flow performance. Several statistical variables have been defined for this very purpose:

- **(N-1)-port PDU traversal count.** Two scalar variables for both input and output containing the number of PDUs transferred through the port in each direction.
- **RMT queue length.** A vector variable documenting number of PDUs in a queue over time.
- **RMT queue drop count.** A scalar variable providing the number of PDUs dropped by a queue.

The RMT module also supports ns2-style tracefile generation. This can be enabled by setting the `tracing` parameter of `relayAndMux` to `true`.

5.4 Sample policy implementations

As Relaying And Multiplexing follows the design principle of separation of mechanism and policy, most of its complexity lies in the policies. Hence, to demonstrate the use of RMT's policy framework, I have implemented a diverse set of simple RMT policies.

- **Scheduling policy**
  - *LongestQFirst.* Pick the queue which contains the most PDUs.
- **Monitoring policy**
  - *REDMonitor.* Used in conjecture with REDDropper; Random Early Detection implementation.
- **MaxQ policy**
  - *ECNMarker.* If queue size $\geq$ threshold, apply ECN marking on new PDUs; if size $\geq$ max, drop.
  - *ReadRateReducer.* If queue size $\geq$ allowed maximum, stop receiving data from input ports.
  - *REDDropper.* Used in conjecture with REDMonitor; Random Early Detection implementation.
  - *TailDrop.* If queue size $\geq$ allowed maximum, drop new PDUs.
  - *UpstreamNotifier.* If queue size $\geq$ allowed maximum, send a notification to the PDU sender.
- **PDU Forwarding policy**
  - *SimpleTable.* A table with {(dstAddr, QoS) $\rightarrow$ port} mappings.

To demonstrate the abilities of Resource Allocator's RMT management, I have also implemented an additional set of management policies (introduced in Section 5.3.4).

- **QueueAlloc**
  - *QueuePerNFlow.* Maintain a queue for each (N)-flow.
  - *QueuePerNQoS.* Maintain a queue for each (N)-QoS cube.
- **QueueIDGen**
  - *IDPerNFlow.* Companion policy for QueueAlloc::QueuePerNFlow.
  - *IDPerNQoS.* Companion policy for QueueAlloc::QueuePerNQoS.

Chapter 6

Testing and Evaluation

I have created a set of basic network topologies to demonstrate the implementation's features. The following tests put them into use. Each test case contains a description of RMT-related events that happen during simulation. Each type of event is described only once to prevent needless repetition.

For purposes of testing, RINASim contains a ping-like application called `AEPing` that can serve as both sender and receiver of a ping-like data frame. Each of the following scenarios consists of a NED topology and two instances of `AEPing` on different hosts. The first instance with application name `App1` allocates a flow to the other instance with application name `App2`, pings the other instance multiple times in sucession and then deallocates the flow. The simulation times of flow allocation, ping send events and flow deallocation are configurable via the `AEPing` NED module.

Additionally, each channel connecting two hosts is configurable to simulate postponed message delivery based on its bandwidth or fixed delay. To keep the format of timestamps simple, channels in the following examples operate with a fixed delay in seconds.

6.1 Basic Multiplexing

This example presents the multiplexing function of RMT in a simple scenario. Coincidentally, it also presents RINASim's implementation of the Allocate algorithm described in Section 3.5.2.

6.1.1 Topology

The TwoCS topology is the most basic example of a computer network. It consists of two interconnected hosts, each with a single application and two levels of IPC processes. Details are depicted on Figure 6.1.

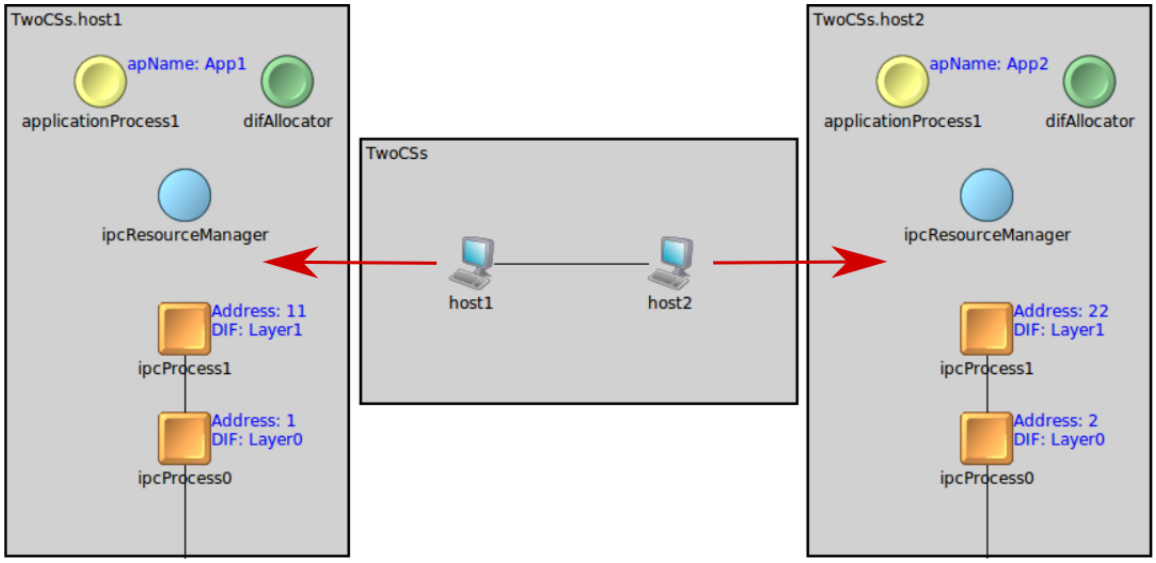


Figure 6.1: TwoCS simulation scenario

6.1.2 Scenario

The configuration of this simulation scenario is described in Table 6.1.

module	configuration directive	value
`host1`'s `AEPing`	flow allocation	10 s
`host1`'s `AEPing`	first ping	35 s
`host1`'s `AEPing`	flow deallocation	200 s
`host1`'s `AEPing`	number of pings	10
`host1`→`host2` channel	delay	2 s

Table 6.1: Configuration for Basic Multiplexing.

6.1.3 Simulation

What follows is a simplified description of events related to RMT instances.

- **t=0**: The RMT instances undergo their initial setup. For the bottom IPC processes in both hosts, this consists of creating a RMTPort instance named `PHY` and its queues. Since the QueueAlloc policy is set to `SingleQueue`, the following queues are created: `outQ_M` (management output), `inQ_M` (management input), `outQ_0` (data output) and `inQ_0` (management input).

- **t=10**: `App1` invokes *Allocate(`App2`)*. This in turn causes IPC `11`'s Resource Allocator to invoke *Allocate(22)*.

  The RIB in IPC `1` sends out an `M_CREATE` request addressed to IPC `2`. RMT stores the PDU to the `PHY`'s queue `outQ_M`.

The arrival of PDU into the queue causes invocation of the active Monitoring policy and MaxQ policy. If the MaxQ policy has not caused the PDU to be dropped, the Scheduling policy is invoked.

The Scheduling policy decides to release the PDU from `outQ_M`. The PDU traverses through the port and gets sent to the other host via the medium.

- **t=12**: IPC `2`'s RMT receives the PDU on port `PHY`. As this is a management PDU, it is stored into the management input queue `inQ_M`.

  The arrival of PDU into the queue causes invocation of the active Monitoring policy and MaxQ policy. If the MaxQ policy has not caused the PDU to be dropped, the Scheduling policy is invoked by the dispatcher algorithm.

  The Scheduling policy decides to release the PDU from `inQ_M`. The PDU travels into the dispatcher which relays the PDU to the RIB daemon.

  The RIB daemon sends out an `M_CREATE_RESPONSE` reply to IPC `1`.

- **t=14**: The IPC `1`'s RIB daemon receives the `M_CREATE_RESPONSE` reply and the *Allocate(22)* procedure is sucesfully completed. The new connection is mapped to a newly instantiated RMTPort called `p0` and the previously suspended *Allocate(App2)* procedure is triggered to continue and carry out the same series of events, only one level higher and using IPC `11`'s port `p0` and its data queues.

- **t=18**: The *Allocate(App2)* procedure is sucessfully completed and `App1` is now connected to `App2`.

- **t=35**: In `host1`, `App1` sends out the first ping. The ping traverses RMTs in IPC `11` and IPC `1` and then it is sent to the medium. This is repeated nine times in the following 9 seconds.

- **t=37**: In host2, IPC `2`'s port `PHY` receives the first ping. The ping traverses RMTs in IPC `2` and IPC `22` and then it is handed to App2. This is repeated nine times in the following 9 seconds.

- **t=200**: App1 invokes *Deallocate(App2)*.

- **t=204**: The connection between `App1` and `App2` is deallocated.

6.1.4 Evaluation

The implementation reflects the behavior expected from RINA specifications. The procedures of the basic communication model described in Section 3.5.2 are executed properly and RMT's multiplexing functionality correctly handles traversing PDUs.

6.2 Basic Relaying

This example presents the relaying function of RMT in a simple scenario.

6.2.1 Topology

The SimpleRelay topology is the most basic example of a computer network with a relaying device. It consists of two hosts interconnected by a router with three IPC processes. More on Figure 6.2.

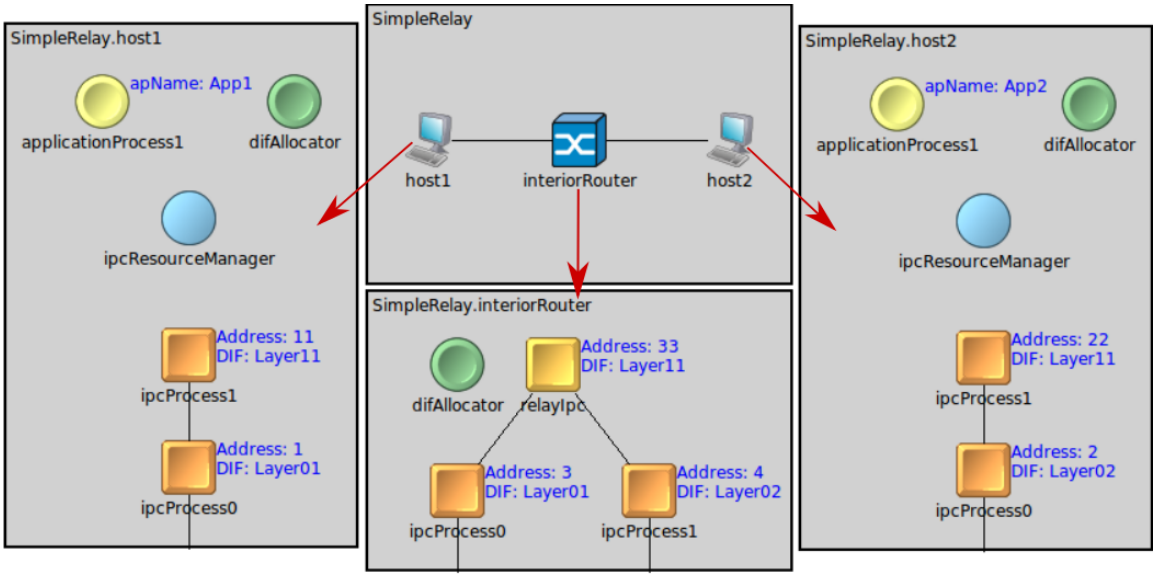


Figure 6.2: TwoCSs network topology.

6.2.2 Scenario

The configuration of this simulation scenario is described in Table 6.2.

module	configuration directive	value
`host1`'s `AEPing`	flow allocation	10 s
`host1`'s `AEPing`	first ping	35 s
`host1`'s `AEPing`	flow deallocation	200 s
`host1`'s `AEPing`	number of pings	10
`host1`→`interiorRouter` channel	delay	2 s
`interiorRouter`→`host2` channel	delay	2 s

Table 6.2: Configuration for Basic Relaying.

6.2.3 Simulation

- **t=10**: `App1` invokes *Allocate(`App2`)*. This in turn causes IPC `11`'s Resource Allocator to invoke *Allocate(`33`)*.
- **t=14**: The IPC `1`'s RIB daemon receives the `M_CREATE_RESPONSE` reply and the *Allocate(IPC `33`)* procedure is sucesfully completed. The new connection is mapped to a newly instantiated RMTPort called `p0` and the previously suspended *Allocate(`App2`)* procedure is triggered to continue. The `M_CREATE` sent by IPC `11`'s RIB daemon is directed to IPC `33`'s RIB daemon.

- **t=16**: The IPC 33's RIB daemon sees that the M_CREATE requires acces to App2, which is not a local aplication, so the request has to be forwarded to IPC 22. It holds the request and waits for Resource Allocator to process *Allocate(22)*.
- **t=20**: The IPC 4's RIB daemon receives the M_CREATE_RESPONSE reply and the *Allocate(IPC 22)* procedure is sucesfully completed. The new connection is mapped to a newly instantiated RMTPort called p1.

  The M_CREATE which was withheld in t=16 then continues forward to IPC 22.
- **t=22**: The IPC 22's RIB daemon receives the M_CREATE request and responds with a M_CREATE_RESPONSE reply, this time directed directly to IPC 11.
- **t=26**: The *Allocate(App2)* procedure is sucessfully completed and App1 is now connected to App2 via interiorRouter.
- **t=35**: In host1, App1 sends out the first ping.
- **t=37**: In IPC 33, the RMT receives the ping PDU from port p0. It discovers the PDU's address does not equal IPC 33, so it looks up the PDU Forwarding Policy for an item with PDU's destination (IPC 22) and PDU's QoS-ID (1).

  The Forwarding policy returns a single RMTPort, p1. The QueueIDGen policy is given the PDU as a parameter and returns outQ_0.

  The dispatcher directs the PDU to p1's queue outQ_0, where it is immediately processed by policies and sent.
- **t=39**: In host2, App2 receives the ping and sends back an acknowledgment message.
- **t=41**: In IPC 33, the RMT dispatcher receives the ping reply PDU from port p1. It looks up the Forwarding Policy again and then sends out the PDU through p0.
- **t=200**: App1 invokes *Deallocate(App2)*.
- **t=208**: The connection between App1 and App2 is deallocated.

6.2.4 Evaluation

The implementation reflects the behavior expected from RINA specifications. The procedures of the basic communication model described in Section 3.5.2 are executed properly and RMT's relaying functionality correctly forwards PDUs.

6.3 Advanced examples

The medium attached to this thesis contains more example simulation scenarios. They are mostly aimed at demonstrating the use of the default policy set described in Section 5.4 and more advanced network topologies.

Chapter 7

Conclusion

In this thesis, I have taken a brief look into the field of network architecture research. I have described the motivations behind research efforts and analyzed several new network architectures.

The implementation goal of this thesis was to contribute to prototyping attempts of one of the presented network architectures, RINA. This task has been sucessfully completed by implementing RINA's Relaying and Multiplexing Task into an existing library for simulation tool OMNeT++. The resulting solution is currently in use by multiple research groups for modelling RINA networks and experimenting with various policies.

7.1 Own Contributions

To be able to describe the principles of new network architectures, I have studied research articles that have been written about them. However, to analyze their usefulness to the Internet, I needed to gain understanding of the driving factors behind their inception. This has been achieved by studying the Internet and its history to learn about choices that led to its current state, including its present problems. I have noticed that some of the problematic design choices of its architectural design have been recognized right at the early beginning of its deployment, but they were eventually chosen as foundations for other then technological reasons.

To understand RINA, I had to learn about its design principles and adapt to its paradigm shift which abandons most of today's widely recognized principles of computer networking. This required extensive studying of John Day's book *Patterns in Network Architecture* [6] and bleeding-edge architectural specifications.

The implementation part of this thesis required me to learn about the OMNeT++ programming framework and the RINASim library. As RMT's specifications were only brief and its other implementations provided only a limited set of functionality, I needed to put some initiative into coming up with the implementation design. This usually involved discussing architectural matters with RINA researchers. In the later phase of development, several of the researchers have raised multiple functionality requests; all of them were eventually implemented.

7.2 Future Development

The implementation of Relaying and Multiplexing Task provides users with a policy framework that allows them to experiment with virtually unlimited number of approaches to the problems of forwarding and congestion avoidance in RINA. Therefore, potential for future development lies mainly in expansion of the policy set.

Several examples of such new policies written by RINA researchers are already available in the GitHub source code repository at the time of writing this thesis. Such examples usually explore new directions in the areas of congestion avoidance, routing and distributed resource allocation.

Bibliography

[1] Saamer Akhshabi and Constantine Dovrolis. "The evolution of layered protocol stacks leads to an hourglass-shaped architecture". In: *ACM SIGCOMM Computer Communication Review*. Vol. 41. 4. ACM. 2011, pp. 206–217.

[2] *Available Pool of Unallocated IPv4 Internet Addresses Now Completely Emptied.* <https://www.icann.org/resources/press-material/release-2011-02-03-en>. [Online; accessed 01-April-2015].

[3] Akash Baid, Tam Vu, and Dipankar Raychaudhuri. "Comparing alternative approaches for networking of named objects in the future internet". In: (2012), pp. 298–303.

[4] S. Bellovin. *Defending Against Sequence Number Attacks*. RFC 1948 (Informational). Obsoleted by RFC 6528. Internet Engineering Task Force, May 1996. URL: <http://www.ietf.org/rfc/rfc1948.txt>.

[5] *BGP Routing Table Analysis Reports.* <http://bgp.potaroo.net/>. [Online; accessed 01-April-2015].

[6] John Day. *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall Pearson Education distributor, 2008.

[7] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460 (Draft Standard). Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112. Internet Engineering Task Force, Dec. 1998. URL: <http://www.ietf.org/rfc/rfc2460.txt>.

[8] K. Egevang and P. Francis. *The IP Network Address Translator (NAT)*. RFC 1631 (Informational). Obsoleted by RFC 3022. Internet Engineering Task Force, May 1994. URL: <http://www.ietf.org/rfc/rfc1631.txt>.

[9] *Future Internet Architectures – Next Phase (FIA-NP).* <https://www.nsf.gov/funding/pgm_summ.jsp?pims_id=504882>. [Online; accessed 12-May-2015].

[10] Ali Ghodsi et al. "Information-centric networking: seeing the forest for the trees". In: *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. ACM. 2011, p. 1.

[11] *iOS: Multipath TCP Support in iOS 7.* <https://support.apple.com/en-us/HT201373>. [Online; accessed 12-May-2015].

[12] *IPv6 Adoption.* <https://www.google.com/intl/en/ipv6/statistics.html>. [Online; accessed 01-April-2015].

[13] James Kurose. *Computer networking : a top-down approach featuring the Internet*. Boston: Addison-Wesley, 2003. ISBN: 978-0201976991.

[14] D. Meyer, L. Zhang, and K. Fall. *Report from the IAB Workshop on Routing and Addressing*. RFC 4984 (Informational). Internet Engineering Task Force, Sept. 2007. URL: <http://www.ietf.org/rfc/rfc4984.txt>.

[15] David Naylor et al. "XIA: architecting a more trustworthy and evolvable internet". In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014), pp. 50–57.

[16] Y. Rekhter, T. Li, and S. Hares. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271 (Draft Standard). Updated by RFCs 6286, 6608, 6793. Internet Engineering Task Force, Jan. 2006. URL: <http://www.ietf.org/rfc/rfc4271.txt>.

[17] J. Saltzer. *On the Naming and Binding of Network Destinations*. RFC 1498 (Informational). Internet Engineering Task Force, Aug. 1993. URL: <http://www.ietf.org/rfc/rfc1498.txt>.

[18] R. Stewart. *Stream Control Transmission Protocol*. RFC 4960 (Proposed Standard). Updated by RFCs 6096, 6335, 7053. Internet Engineering Task Force, Sept. 2007. URL: <http://www.ietf.org/rfc/rfc4960.txt>.

[19] Eleni Trouva et al. "IS THE INTERNET AN UNFINISHED DEMO? MEET RINA!" In: (2010).

[20] Arun Venkataramani et al. "MobilityFirst: a mobility-centric and trustworthy internet architecture". In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014), pp. 74–80.

[21] Richard W Watson. "The Delta-T transport protocol: Features and experience". In: *Local Computer Networks, 1989., Proceedings 14th Conference on*. IEEE. 1989, pp. 399–407.

[22] *What caused today's Internet hiccup*. <https://www.bgpmon.net/what-caused-todays-internet-hiccup/>. [Online; accessed 12-May-2015].

[23] Lixia Zhang et al. "Named data networking (ndn) project". In: *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC* (2010).

Appendix A

CD Contents

The attached CD contains the following files and folders:

- `thesis.pdf`, a PDF version of this thesis.
- `tex/` containing the LaTeX source codes of this thesis.
- `src/` containing the April release of the RINASim library.
  - `DIF/RMT/` containing the RMT source codes.
  - `DIF/RA/` containing the RA source codes.
- `examples/` containing the set of examples used in Chapter 6 along with more advanced ones.

Appendix B

Class Diagram

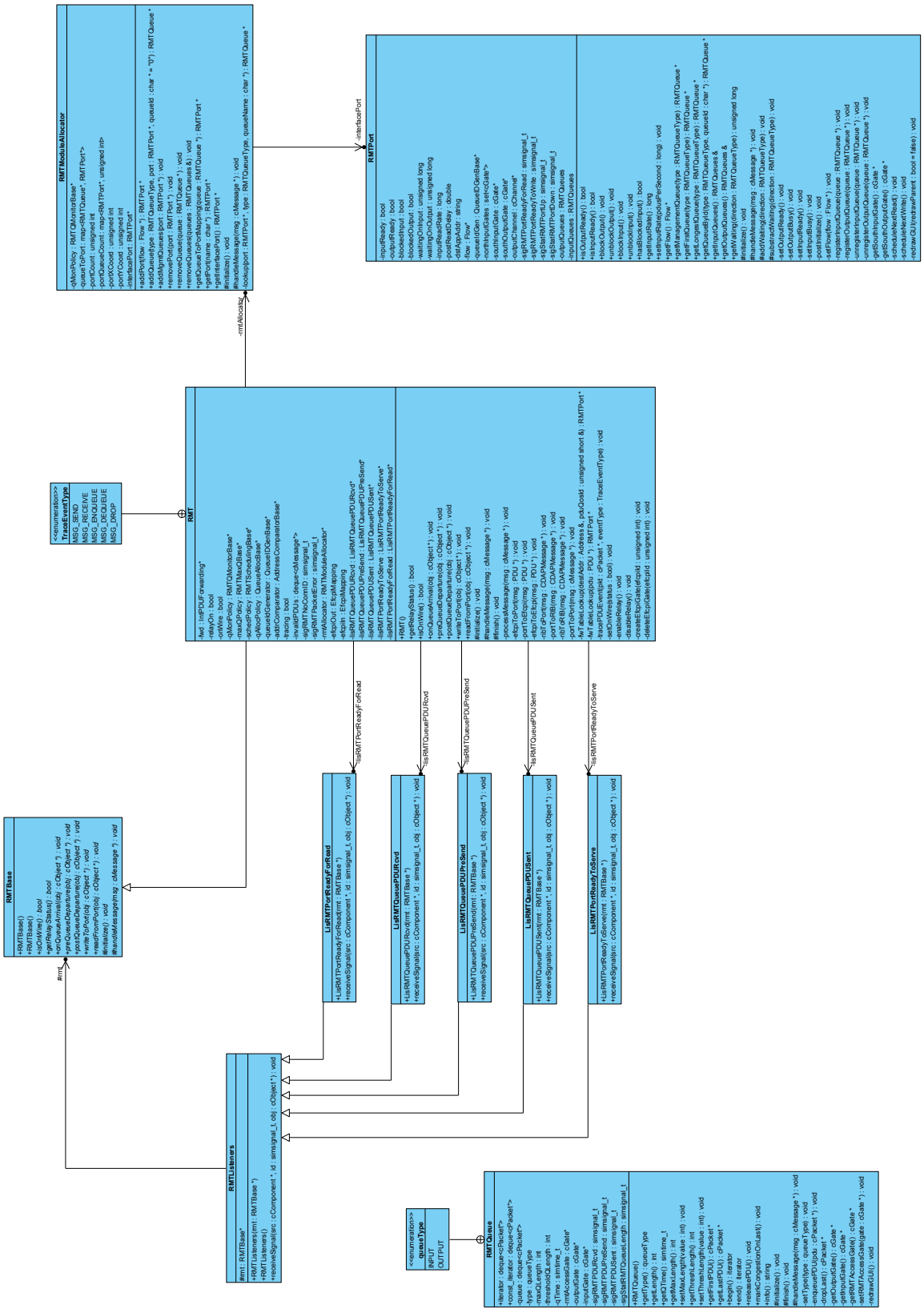


Figure B.1: RMT class diagram.