VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODELLING NEW NETWORK ARCHITECTURES IN OMNET++

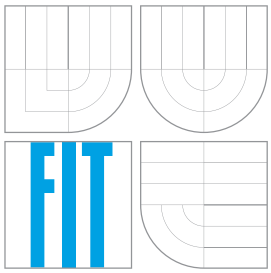
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ HYKEL

BRNO 2015

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 některých nových síťových architektur. Dále je dokumentována implementace modulu Relaying and Multiplexing Task síťové architektury RINA pro simulační framework OMNeT++. Cílem práce je doplnění funkcionality stávající simulační knihovny pro zajištění plnohodnotného modelování sítí RINA. Cílem práce je umožnit modelování daných protokolů bez nutnosti budování fyzické architektury.

Abstract

This thesis describes principles and state of implementation of some new network architectures. It also documents implementation of a significant part of one of the presented architectures, RINA, for the OMNeT++ simulation framework. The main goal of this thesis is to extend functionality of an existing simulation library to provide full-fledged means of modelling RINA networks.

Klíčová slova

OMNeT++, RINA, RMT, počítačové sítě, síťové architektury

Keywords

OMNeT++, RINA, RMT, 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 11, 2015

Poděkování

:*

© Tomáš Hykel, 2015.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today's field of computer networking and its research is heavily centered around the underlying architecture of the Internet and its TCP/IP protocol suite. While such concepts are in use for several decades and seem to work as intended, there has been a growing trend in the research community of introducing new alternative network architectures. This thesis aims to illustrate aims 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. omnet

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 the Internet. 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 significant part of one of the presented architectures, RINA, for the OMNeT++ discrete simulation 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 later chapters as a reference point for evaluating contributions of alternative architectures.

Chapter 3 presents an outline of current projects in field of network architecture and describes a representative set of architectures.

Chapter 4 sums up and evaluates contributions brought by the previously described architectures.

Chapter 5 takes a closer look at parts of Recursive InterNetwork Architecture that are related to the implementation.

Chapter 6 describes implementation of RINA's Relaying and Multiplexing Task for OMNeT++.

Chapter 7 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 keeps growing even 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, when the the demands on internetworking capabilities were nowhere compared to now.

During the Internet's growth, whenever there has been a problem that required a solution, it's 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. This way of improving the Internet's base technology is convenient since each paradigm shift in foundations of the Internet can require a long and expensive transfer of existing network configurations to the new technology. The most notable example of this is the internet layer protocol IPv6 requiring explicit firmware support from active network components. The problem of IPv4 space exhaustion has been known of since the beginning of 1990s, the first formal IPv6 specification arose in 1996 (RFC 2460) and the first IPv6 routers emerged in 2004 – and yet, as of 2015, two years after the top-level IPv4 pool exhaustion, 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 [3].

As such, some of the Internet's widely recognized problems are inherent because of the base design and it's 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, Jerry Saltzer in his work "On the Naming and Binding of Network Destinations" [6] 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 2.1. At the time, network architectures such as CYCLADES, XNS, DECNET and OSI conformed to this scheme [7].

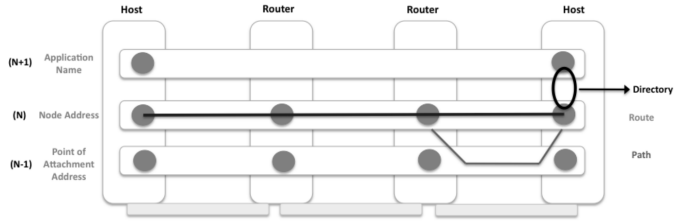


Figure 2.1: Jerome Saltzer's view of computer networking

TCP/IP doesn't 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 even 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. 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 (section 2.2) and mobility (section 2.3). Borrowing a comparison from John Day's Patterns in Network Architecture, the lack of implicit logical addressing in the Internet "feels like accessing computer memory with DOS instead of Unix".

2.2 Lack of Multihoming

Since IP addresses serve as point-of-attachment addresses (i.e. an IP address identifies a system interface) and routing is done exclusively on IP's layer, there isn't any inherent mechanism for distinguishing whether multiple IP addresses identify a common node. The insufficient base for multihoming support is also one of the oldest recognized problems of the Internet: it has become 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 [2]. In spite of this, the switch to a new protocol suite that happened 11 years later (on 1.1.1983, the "flag day") didn't 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 SCTP provides a partial support. Two SCTP hosts are able to provide each other with lists of alternate IP addresses that can be used as fallback points of attachment of the same application in case of failure of the primary address. During such session, each SCTP host needs to continually check other host's endpoints by heartbeat packets to make sure they are accessible via other sessions. However, thus far, multiple reasons have been preventing SCTP from becoming a widely known and used solution; the biggest disadvantage lies in the fact

that due to TCP/IP not having 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 sockets 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, which provides means for load-balancing and fall-back 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 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.

2.3 Lack of Mobility

Since a host location is determined by its IP address and IP addressing is location-dependent, mobility is essentialy non-existent.

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

- **Mobility by indirection.** A fixed host 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 MobileIP, LISP or i3.

- **Global name resolution.** Each packet contains an identifier which . TODO This approach is used by technologies such as XIA or MobilityFirst.

- **Name-based routing.** TODO This approach is used by technologies such as NDN or TRIAD.

The second and third approaches require a clean-slate architectural design and can be seen in some of the architectures presented in this thesis.

2.4 Lack of security mechanisms

The specifications of the fundamental protocols of TCP/IP stack – IP, TCP and DHCP – were originally finished 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).

TODO: lack of verifiable identifiers

In addition to that, a large amount of security flaws has been discovered and continually exploited. Some examples of these are Denial-of-service attacks (ICMP flood, SYN

flood, CAM flood), Man-in-the-Middle attacks, IP address spoofing, DHCP spoofing, ARP spoofing and ARP cache poisoning. The requirement of every secure network is to carefully mitigate all of them.

2.5 Router Table Size Growth

Default-free zone (DFZ) is the collection of all Internet autonomous systems (AS) that do not require a default route to route a packet to any destination. Since they comprise the root of the 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.

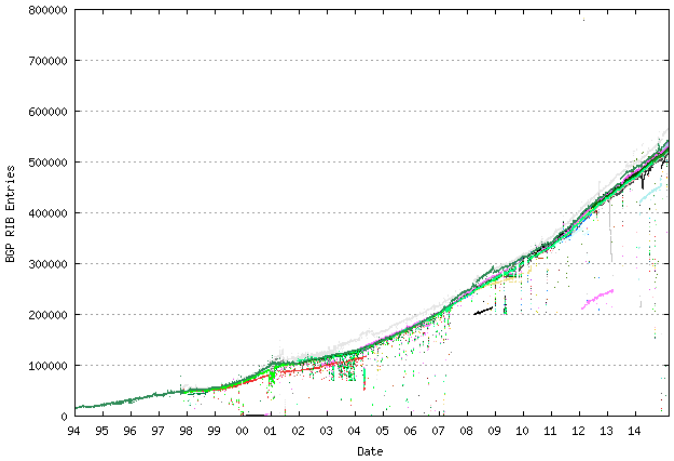


Figure 2.2: Global Internet routing table size growth [1]

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

As of year 2015, the Internet routing table is consisted of over 560k entries, which requires over 0.5 GB of memory on each router. It's been commonly believed that Moore's law will ensure that the technology of high-end routers will keep scaling along with the increased demands, but recent research proves otherwise.

Chapter 3

Alternative Network Architectures

This chapter gives an overview of paths that were pursued in the field of network architecture research and descriptions of several architectures.

Considering the limited scope of this thesis, only a representative subset of new network architectures will be described and evaluated more thoroughly. The subset consists of projects emerged under the Future Network Architecture (Named Data Networking [3.3], MobilityFirst [3.4], XIA [3.5] and NEBULA [3.6]) and the Recursive InterNetwork Architecture (3.7).

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. SDN, LISP, DiffServ, IntServ), or
- **clean slate design**: designing completely new standalone architectures that aren't constrained by Internet technology's limitations (e.g. NDN, RINA)

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

3.2 Research Initiatives

The idea of funding research for exploring new architecture has been prevalent especially since the year 2000 and NewArch project. It's common to group such research initiatives under a common name "Future Internet".

TODO: mention testbeds; describe worldwide research programs or fuck it? only FIA and possibly John Day are relevant for us

3.2.1 Future Internet Architecture

In 2010, National Science Foundation funded five projects as a part of this program: Named Data Networking, MobilityFirst, NEBULA, eXpressive Internet Architecture and ChoiceNet.

3.3 Named Data Networking

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

3.3.1 Premise

TODO: the thin waist stuff is referred to by other archs as well, maybe move it somewhere else?

During its early ears, ARPANET was heavily influenced by telecommunication technology as the PSTN was the only example a large-scale network. Due to this, technology of the Internet has been built on the paradigm of host-to-host communication driven by hosts' addresses. This dependence is often visually presented on an "hourglass model", which indicates that while there's a wide array of technologies in use in the lower and higher layers of the Internet technology stack, the hourglass's "thin waist" of end-to-end communication exercised by the IP protocol is the key part common for all networks.

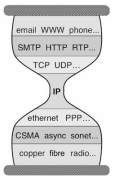


Figure 3.1: Internet's "thin waist"

While the base paradigm of host-to-host communication has remained constant over the decades, the way we use the Internet has considerably changed: the Internet has become mostly a content distribution network. Since the mechanism of communication over the Internet is based on creating and maintaining end-to-end connections, this creates 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.

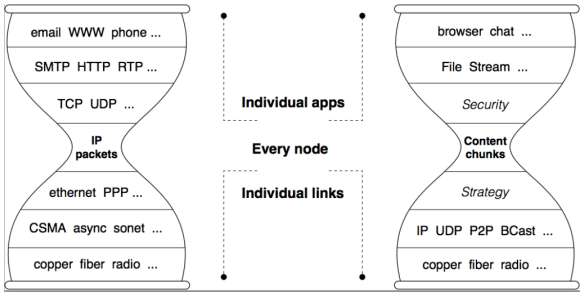


Figure 3.2: NDN's "thin waist"

3.3.2 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)
- two types of hosts: a consumer (data requester) and a producer (data provider)
- a router maintaining three fundamental data structures:
  - Forwarding Information Base (forwarding table)
  - Pending Interest Table (maintaining currently active requests)
  - Content Store (data cache)

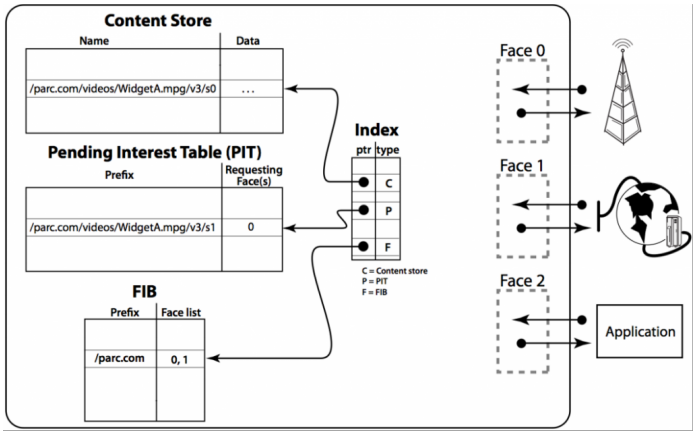


Figure 3.3: NDN router

The Communication Model

Communication in NDN is driven by the data receiver, i.e. the consumer.

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 Pending Interest Table.
     - 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 either 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 isn't defined by the architecture itself and should be a subject of further research.

3.3.3 Current State of Implementation

NDN's implementation efforts are open-source and available as a package called `NDN Platform` [4]. 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` [5], based on `ns-3`, is a network simulator using `ndn-cxx` and `NFD`.

3.4 MobilityFirst

3.4.1 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 means for better mobility, while also introducing intristic security properties and faciliating services.

3.4.2 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.

Loc/ID Separation

MobilityFirst's "narrow 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 today's autonomous sytems.

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 to 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 Domain Name System 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 via 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.

3.4.3 Current State of Implementation

- 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, other based on OpenFlow

3.5 eXpressive Internet Architecture

3.5.1 Premise

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

3.5.2 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, the processing that is required to forward traffic with addresses of its type and the intristic security properties. The initial XIA architecture defines four basic XIA principal types:

- Host XIDs: 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 XIDs: SIDs support communication with (typically replicated) services and realize anycast forwarding scheme. SIDs define what entities do.
- Content XIDs: 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 XIDs: 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 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.

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 (3.4a). However, in addition to that, they can also contain scoping information (e.g. target network + a service located in the network [3.4b]) or fallback alternatives (e.g. Network XID to be used by forwarding when given SID isn't recognized [3.4c]).

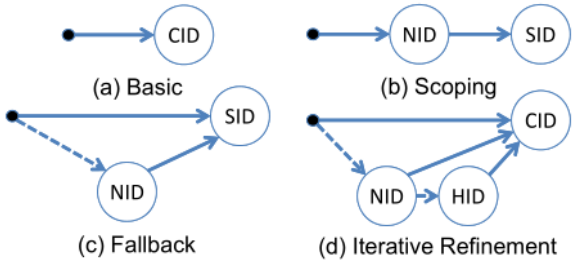


Figure 3.4: DAG-based addressing in XIA

3.5.3 Current State of Implementation

- a Click-based prototype, including the XIA protocol stack, routing, nameserver, API
- a native Linux network stack implementation with attempts to port different architectures to XIA

3.6 NEBULA

3.6.1 Premise

NEBULA is based on a cloud computing-centric vision of a future Internet consisting of multiple reliable data centers.

What is missing is a network architecture, which needs to both interconnect data centers and connect users to their data. The current Internet isn't by nature suited for this due to its best-effort delivery model. The needed properties are illustrated in NEBULA papers on a concrete example of closed-loop control of blood glucose levels for an insulin-dependent diabetic:

"The cloud application would determine the current glucose level, monitor what was being eaten, monitor the exercise activity level, and make an insulin infusion recommendation. This application would have strict confidentiality requirements (it is very personal healthcare data), integrity requirements (incorrect dosages can be harmful) and availability requirements (a lack of network availability might, at the limit, cause physical harm)."

NEBULA is a network architecture that aims to provide all properties necessary for global cloud computing.

3.6.2 Concepts

NEBULA specifies three building blocks: a reliable routing system and data center interconnect (NCore), a data plane supporting policy enforcement (NEBULA Data Plane, NDP) and a control plane (NEBULA Virtual and Extensible Networking Techniques, NVENT)

NCore

NDP

NVENT

3.6.3 Current State of Implementation

3.7 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.

3.7.1 Premise

RINA introduces a novel approach to the problem of computer networking, eliminating all problems described in chapter 2 in the process:

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

3.7.2 Concepts

RINA specifications define a rich set of new concepts based mostly on the theory of distributed computing.

DAF

Distributed Application Facility (DAF) is a collection of two or more cooperating application processes in one or more computing systems, which exchange information using IPC and maintain shared state. In some Distributed Applications, all members will be the same, i.e. a homogeneous DAF, or may be different, a heterogeneous DAF.

DIF

Distributed IPC Facility is a collection of two or more application processes cooperating to provide InterProcess Communication (IPC). A DIF is a DAF that does IPC. The DIF provides IPC services to Application Processes or IPC Processes of other DIFs via a set of API primitives that are used to exchange information with the Application's peer.

DIF is essentially the RINA's equivalent of a layer. A computing system may be a member of 0-N DIFs and has a separate IPC process with an address for each DIF. A typical computer internetwork would comprise of 3 levels of DIFs, each (N)-DIF handling the data coming from the (N+1)-DIF in the same way as from an application; other DIFs could be added on top in hosts for creating another scope of communication (e.g. for VPN-like facilities).

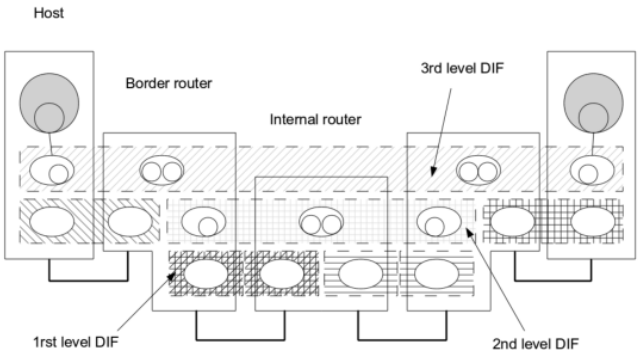


Figure 3.5: An example of RINA internetwork with 3 levels of DIFs

Each DIF operates in its own scope isolated from other DIFs of the same level. Hence, a 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 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 data transfer and IPC management. The behaviour of each component of the IPC process can be configured via policy.

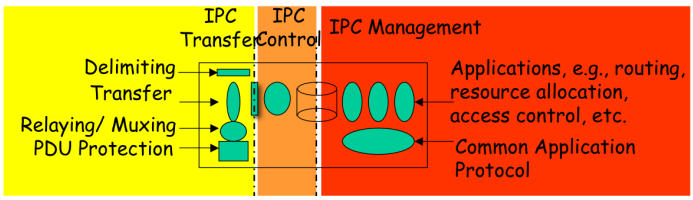


Figure 3.6: Parts of RINA's IPC process

- IPC transfer consists of Delimiting (fragmentation/combination of incoming/outgoing (N+1)-SDU), Error and Flow Control Protocol instances (a Delta-T based protocol handling individual data flows) and Relaying and Multiplexing Task (multiplexing of data from/to EFCP instances/(N-1)-DIFs and optional relaying between (N-1)-IPC processes)
- IPC data transfer is an optional mechanism handling flow control (for example, TCP-like flow control could be implemented here via appropriate policies)
- IPC management handles management tasks such as routing, resource allocation or access control

Addressing

Since each DAF works with its own address scope, the architecture needs to provide means of resolving (N)-application names to addresses of (N-1)-IPC processes. This is achieved via 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").

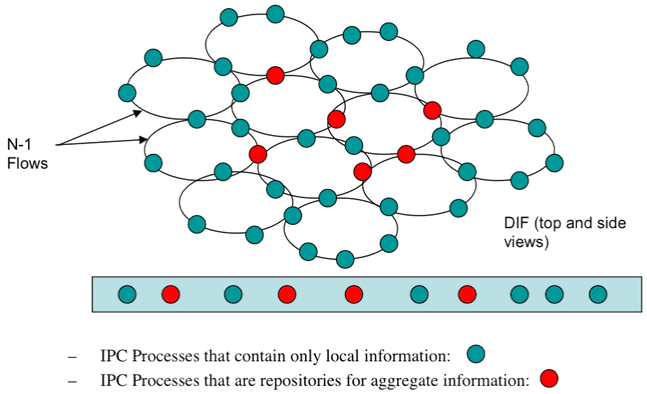


Figure 3.7: Distributed mapping of applications to addresses in a RINA network

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 protocl (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.

Thanks to 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 and provides means 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).

3.7.3 Current State of Implementation

ProtoRINA

IRATI stack

RINASim, an open-source OMNeT++ implementation of RINA, is being actively developed on FIT BUT.

Chapter 4

Architecture Comparison

a	Multihoming	Mobility	Security
NDN	✓	✗	a
MobilityFirst	a	a	a
XIA	a	a	a
NEBULA	a	a	a
RINA	a	a	a

Listing 4.1: Summary of contributions of examined architectures

Chapter 5

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.

5.1 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's 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.

5.2 Relaying and Multiplexing Task

5.2.1 Formal description

Relaying and Multiplexing Task (RMT) has, as its name suggests, two main reponsibilities: relaying and multiplexing of PDUs. The goal of multiplexing is to simply pass PDUs from EFCP instances and RIB to the appropriate (N-1)-flows (and reverse of that). Relaying

handles incoming PDUs from (N-1)-ports[1] that aren't directed to its IPC process and forwards them to other (N-1)-ports using information provided by its forwarding policy.

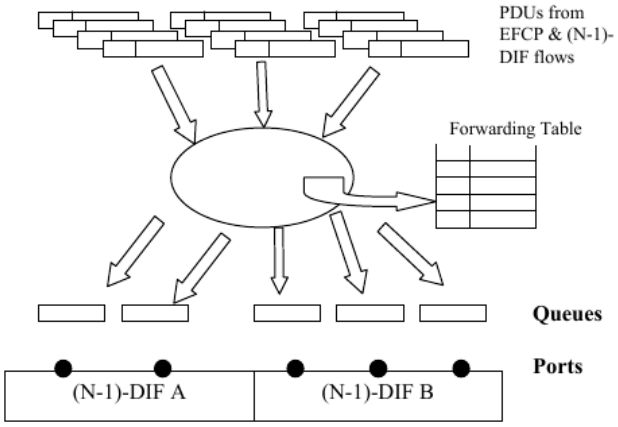


Figure 5.1: Conceptual diagram of Relaying and Multiplexing Task

RMT instances in hosts and bottom layers of routers usually perform only the multiplexing task, while RMTs in top layers of interior/border routers do both multiplexing and relaying. In addition to that, RMTs in top layers of border routers perform flow aggregation.

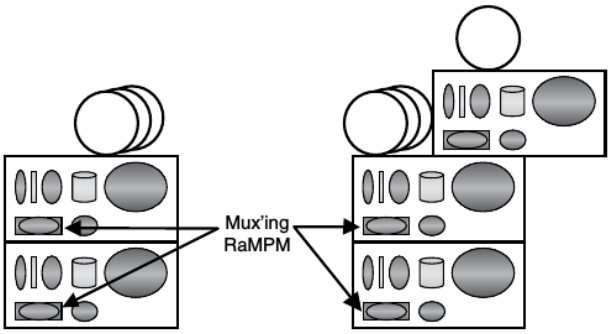


Figure 5.2: A host with multiplexing RMTs

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

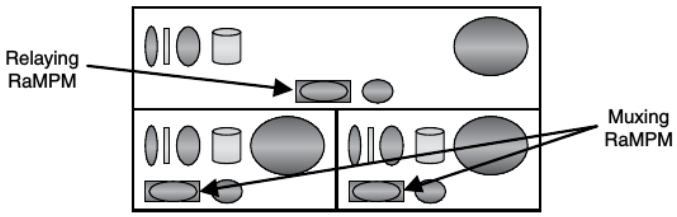


Figure 5.3: An interior router with two multiplexing RMTs and one relaying RMT

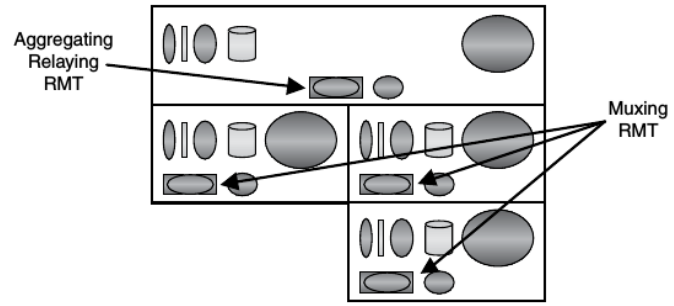


Figure 5.4: A border router with an aggregating RMT

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.

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.

5.2.2 Policies

Even though 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.

- RMT scheduling policy. A scheduling algorithm (also commonly known as "network scheduler algorithm" or "queuing discipline") that determines the order 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 Queuing.

- RMT 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.

- RMT max queue policy. An algorithm that is invoked each time the number of PDUs waiting in a queue exceeds the queue's threshold. This policy is mostly used for implementing congestion control 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 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 6

Implementation of Relaying and Multiplexing Task

6.1 OMNeT++

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

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

6.2 RINASim

RINASim, developed by 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 building RINA networks and observing their behavior. In the current stage of initial 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 MIT licence and publicly hosted by GitHub.

6.3 Process of development

Since RINA is still an emerging architecture, speficications for some of its parts (e.g. Resource Allocator) are still being actively worked on and the current implementations aren't yet fit for production purposes. Hence, there wasn't much to rely on when implementing RMT: the specifications were brief, implementation design non-existent and the only existing implementations were providing only a small subset of RMT's functions. Because of this, most of the later phase of development was driven by PRISTINE researchers' feature requests and frequent mutual feedback regarding architectural or implementational issues.

6.4 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.

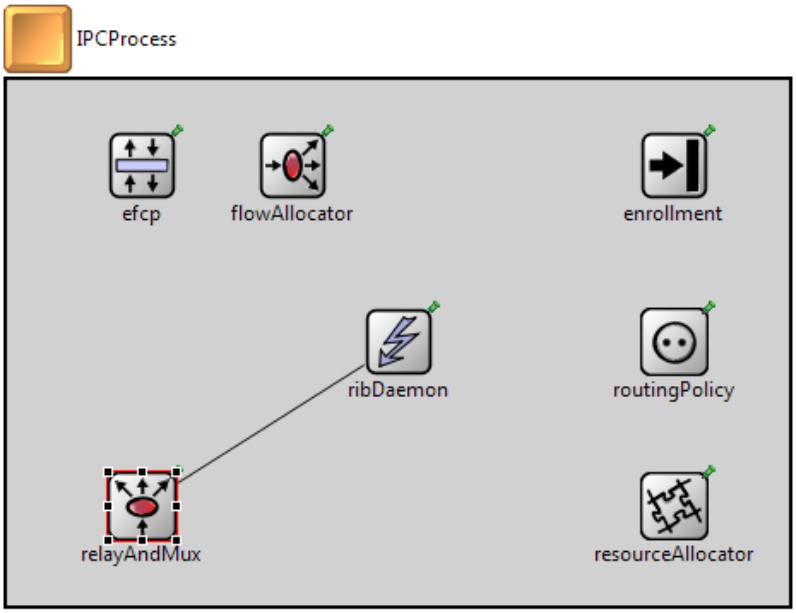


Figure 6.1: RINASim's IPC process

6.4.1 Module Structure

RelayAndMux consists of multiple simple modules of various types, some of which are instantiated only dynamically at runtime.

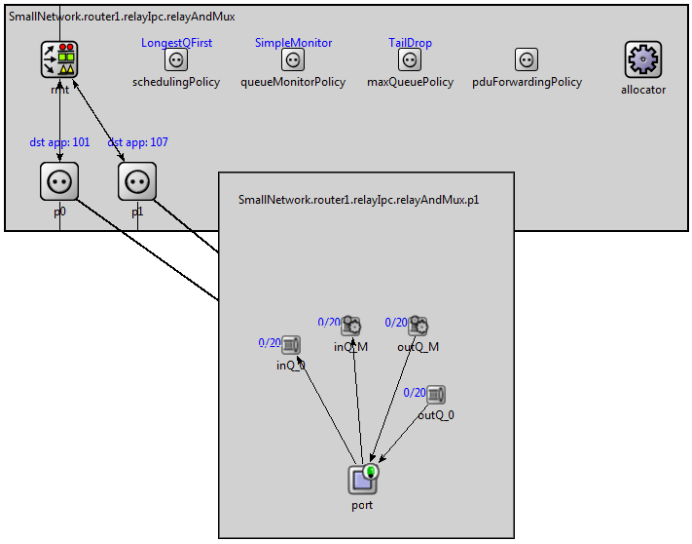


Figure 6.2: RelayAndMux module contents

Static modules

- RMT, the central logic of Relaying And Multiplexing task that decides what should be done with messages passing through the module.
- RMTModuleAllocator, a control unit providing an API for adding and deleting instances of dynamic modules (RMTQueue, RMTPort).
- SchedulingPolicy, a scheduling policy of an RMT instance.
- MonitorPolicy, a monitoring policy of an RMT instance.
- MaxQPolicy, a maxqueue policy of an RMT instance.

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.

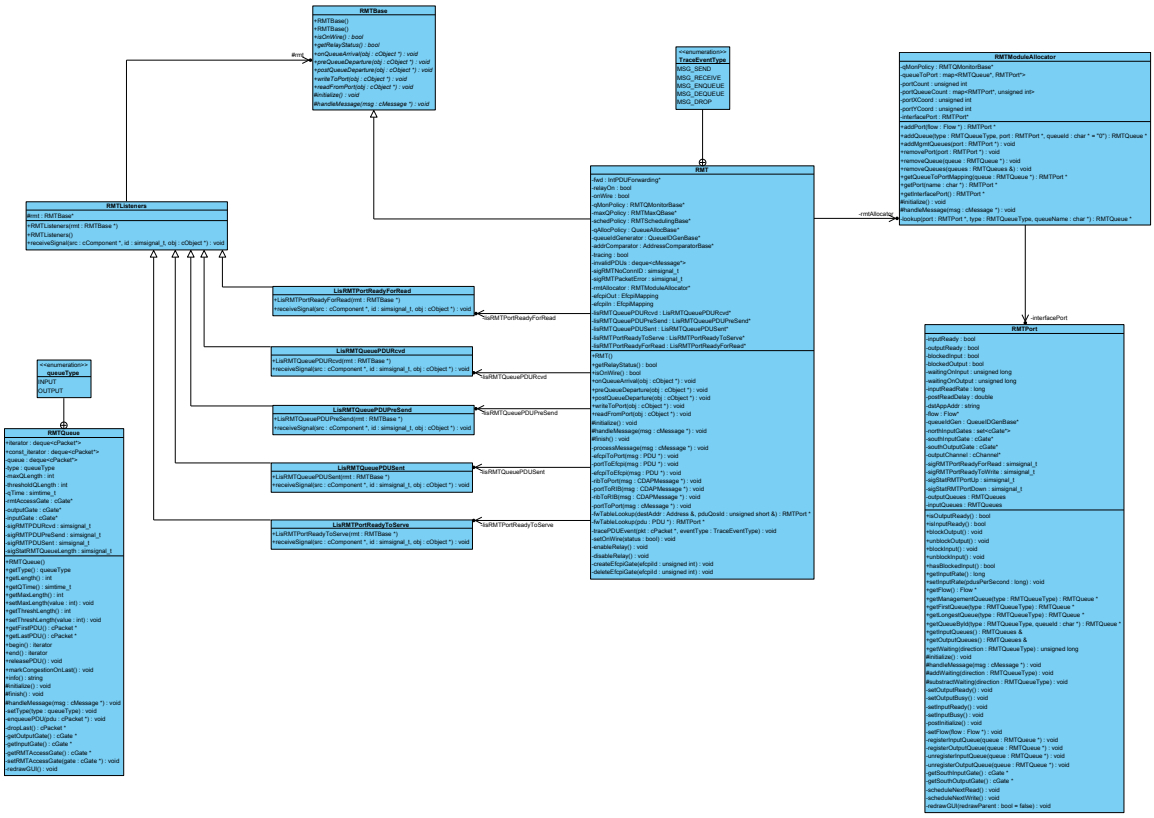


Figure 6.3: RMT class diagram

6.4.2 Module Parameters

Some of the modules contain user-configurable parameters that can be used to alter their behavior.

- *string schedPolicyName*, name of the desired scheduling policy,
- *string qMonitorPolicyName*, name of the desired monitoring policy,
- *string maxQPolicyName*, name of the desired max queue policy,
- *string ForwardingPolicyName*, name of the desired forwarding policy,
- *int defaultMaxQLength*, default maximum length of instantiated queues,
- *int defaultThreshQLength*, default threshold length of instantiated queues, and
- *bool pduTracing*, a switch for turning on PDU tracefile generation.

6.4.3 Module Workflow

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

The decision algorithm decides what to do with the incoming PDU and executes appropriate policies. A state diagram representation can be seen in 6.4. It's implemented

by `handleMessage()` and methods that are subsequently called from it (`portToEFCPI()`, `portToPort()` etc.).

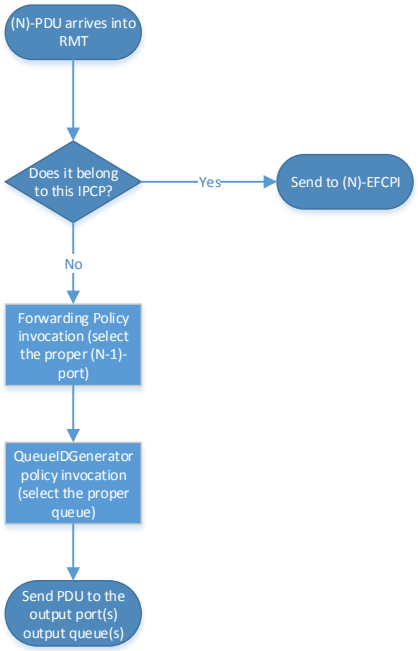


Figure 6.4: The core RMT decision process

The scheduling algorithm ensures continuous invocation of the scheduling policy whenever there are PDUs waiting in queues. A petri net representation can be seen in 6.5. The decision of what queue should be processed next is left to the policy. The algorithm is implemented by methods `onQueueArrival()`, `postQueueDeparture()`, `writeToPort()` and `readToPort()`.

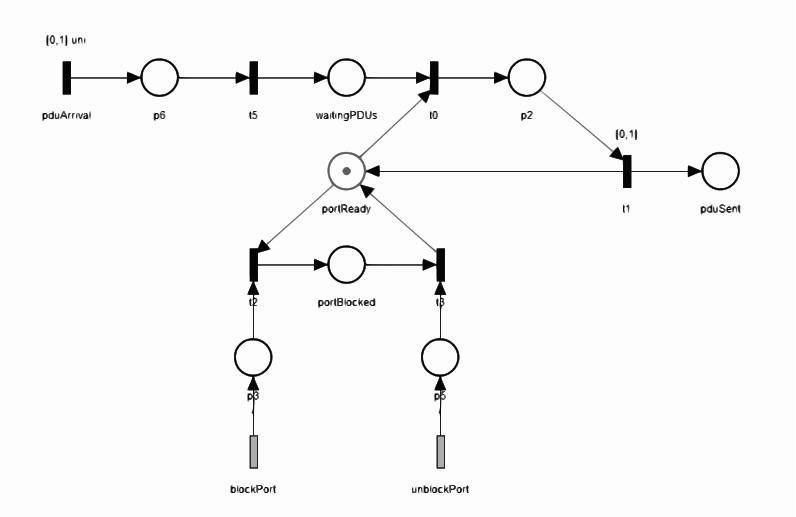


Figure 6.5: The RMT scheduling procedure

6.4.4 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 more concrete specifications at the time of implementation, I have designed a set of management functions, some of which are customizable by policies.

Initial RMT Mode Setup

When a RMT instance is located inside a bottommost IPC process that doesn't work with any further (N-1)-IPC processes, the RMT is switched to an "onWire" mode that functions over a simple medium instead of an IPC connection. Additionally, if a RMT 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()`.

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 creation & removal

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

(N-1)-port data rate control

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.

6.4.5 Statistics Collection

OMNeT++ modules provide 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's 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's 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.

Multiple examples of graphs generated from the variables can be seen in chapter 7.

6.4.6 PDU tracing

6.5 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.

- **Max queue 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 6.4.4).

- **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 7

Testing and Evaluation

A set of basic network simulation scenarios has been created to demonstrate the RMT implementation's features.

7.1 TwoCSs

Two interconnected hosts; host1 sends a series of ping requests to host2.

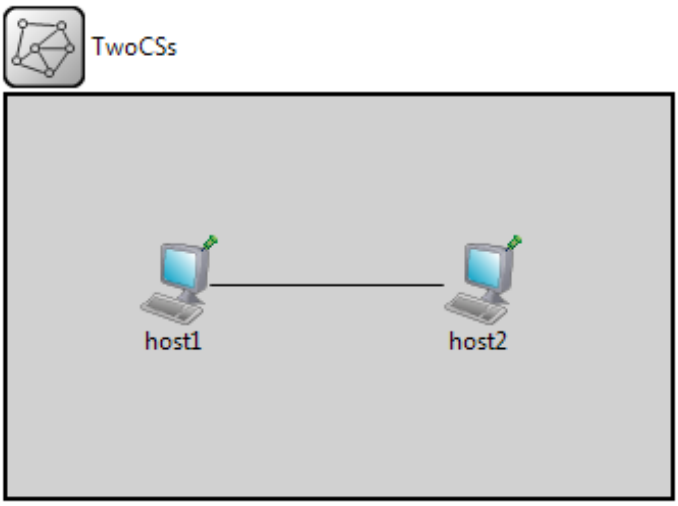


Figure 7.1: TwoCS simulation scenario

7.2 SimpleRelay

Two hosts interconnected by a single router; host1 sends a series of ping requests to host2.

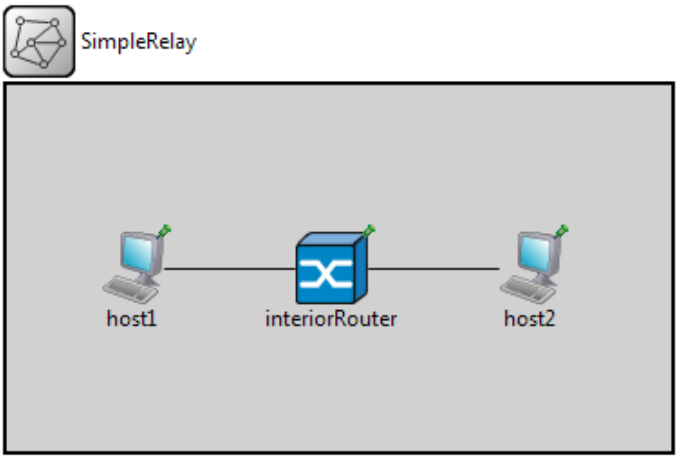


Figure 7.2: SimpleRelay simulation scenario

7.3 SmallNetwork

Five hosts interconnected by three routers; host1 sends a series of ping requests to host5.

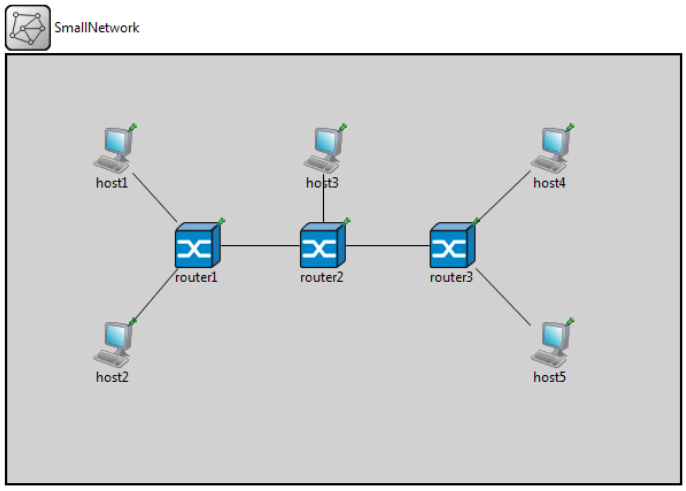


Figure 7.3: SmallNetwork simulation scenario

The output of this scenario can be seen in attachment C. It consists of a simplified simulation log (C.1), graphs generated by Scave (C.2) and a PDU trace file (C.3).

Chapter 8

Conclusion & Future Development

The Relaying and Multiplexing Task has been successfuly implemented and integrated into the RINASim simulation library. The result is currently in use by multiple research groups for modelling RINA networks and experimenting with various RMT policies.

The resulting implementation provides users with a policy framework that allows them to experiment with virtually unlimited number of approaches to forwarding and congestion control in RINA. Therefore, 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 source code repository at the time of writing this thesis.

Bibliography

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

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

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

[4] *NDN Platform*. http://named-data.net/codebase/platform/. [Online; accessed 01-April-2015].

[5] *ndnSIM 2.0*. http://ndnsim.net/2.0/. [Online; accessed 01-April-2015].

[6] 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.

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

Appendix A

CD Contents

The attached CD contains the source code of the whole RINASim library containing the implementation. The library is also equipped with example simulation scenarios.

Appendix B

Test outputs