A GRASP-based Algorithm for the Optimized DIF Allocation in the RINA Network Architecture

Submitted in fulfillment of the requirements for the degree of European Master of Science in Information and Communication Technologies (MINT)

Author: Pegah Keshavarz

Supervisor: Prof. Davide Coreglio
Departament d'Arquitectura de Computadors

Co-Supervisor: Prof. Jordi Perello Muntan
Departament d'Arquitectura de Computadors

December 2013
## Contents:

- **Acknowledgement**: vii
- **Abstract**: viii
- **Chapter 1**: 1
  - 1.1 Objective of final project: 2
  - 1.2. Overview of the thesis: 3
- **Chapter 2**: 4
  - 2.1. Some history: 6
  - 2.2 The absent of internet today: 6
  - 2.3. Some question about internet today: 8
  - 2.4. Defining the Problem Space: 9
    - 2.4.1. What is Network architecture?: 9
    - 2.4.2. Example: the Internet Architecture: 10
    - 2.4.3. Original Requirements: 12
  - 2.5. Architecture of current internet: 13
  - 2.6. Challenges that current internet is facing nowadays: 15
    - 2.6.1. Security and authentication: 15
    - 2.6.2. Availability: 16
    - 2.6.3. Mobility: 16
    - 2.6.4. Cost: 18
    - 2.6.5. Naming: 18
    - 2.6.6. Quality of service and the best effort: 20
  - 2.7. Summary: 21
- **Chapter 3**: 22
  - RINA: 22
    - 3.1. Main things about RINA: 24
      - 3.1.1. Some Definitions: 25
3.2.1 IPC API is in the client application side which will do 4 tasks........................................27
3.2.2. IPC DATA TRANSFER (it is optional).................................................................................28
3.2.3. IPC Management ......................................................................................................................29
3.2.4. An example of one communication in RINA ..........................................................................29
3.3. The content of DIF. .......................................................................................................................30
3.4. IDD: (inter-DIF Directory) ---- distributed application in DIF---- manage a namespace ..............31
   3.4.1. How it works: ............................................................................................................................32
   3.4.3 Information in IDD: ..................................................................................................................33
   3.4.4. IDD DAF as an example ........................................................................................................34
3.5. Some characteristics of RINA and some comparison to TCP/IP ..............................................36
   3.5.1. Naming and addressing in RINA ............................................................................................36
   3.5.2. Mobility and multi-homing in RINA.......................................................................................37
   3.5.3. An example of Hand-off in RINA: ..........................................................................................38
   3.5.4. Security in RINA ....................................................................................................................40
3.6. Summary ........................................................................................................................................41
Chapter 4 ............................................................................................................................................43
Graph Theory & GRASP algorithm: ....................................................................................................43
   4.1. What is graph theory? ..................................................................................................................44
   4.2. Greedy Randomized Adaptive Search Procedure-GRASP .......................................................45
      4.2.1 Introduction to GRASP .........................................................................................................45
      4.2.2. What is GRASP? ..................................................................................................................45
   4.3. Summary .....................................................................................................................................49
Chapter 5 ............................................................................................................................................50
GRASP and Graph theory in RINA: ....................................................................................................50
   5.1. Some Definition: .........................................................................................................................51
   5.2. GRASP-based re-optimization: ...................................................................................................51
      5.2.1. Phase I. Construction phase: .................................................................................................53
      5.2.2. Performing Local Search on the solution space: ...............................................................54
      5.2.3. The Objective function: ......................................................................................................56
   5.3. Summary ....................................................................................................................................58
Chapter 6 ............................................................................................................................................59
Results of the simulation: .....................................................................................................................59
6.1. The parameters and the results: ................................................................. 60
6.2. Analyze based on the number of demands: ........................................ 62
6.3. Analyze base on the different number of iterations: ....................... 63
6.4. Analyze based on the different value of α: ......................................... 65
6.5. Analyze based on the number of IPC processes: ............................. 67
6.6. Summary: ............................................................................................... 70

Chapter 7 ........................................................................................................... 71
Conclusion and future works ........................................................................... 71
  7.1. Conclusion and an overview on our thesis ........................................ 72
  7.2. Future work .......................................................................................... 72
Appendix (a) ..................................................................................................... 74
Dijkstra Algorithm: ......................................................................................... 74
Appendix (b) ..................................................................................................... 82
Codes of simulating .......................................................................................... 82
Bibliography: .................................................................................................... 89
List of figures:

Figure 1.1. Topology of Internet © iStockPhoto / Internet Society .......................... 1
Figure 2.1. The history of TCP/IP ................................................................................ 4
Figure 2.3. Decades of Internet .................................................................................. 8
Figure 2.4. What happens in an internet minute? ...................................................... 12
Figure 2.5. Layers of the current internet architecture ............................................. 14
Figure 2.6. Saltzer’s point of view for a complete naming and addressing schema ................................................................. 19
Figure 3.1. RINA label .............................................................................................. 22
Figure 3.2 An example of RINA architecture with 3-level of DIFs ........................... 24
Figure 3.3. Components of an IPC process ................................................................. 26
Figure 3.4. connection and exchanging information between 2 applications. ....... 28
Figure 3.5. IPC between two application processes in different system ............... 29
3.6 Figure. possible schemas of DIF ..................................................................... 31
Figure 3.10. An example of the IDD DAF in which the search during the first phase of the IDD function is organized according to the hierarchical organization of the application namespace. Neighbor IDD’s are connected to each other (in reality a hop in the IDD graph may be several hops away) ............................................. 35
Figure. 3.10. The forwarding between IDD DAPs during the first phase of the IDD functionality and the relevant search and Neighbor tables contents of each IDD DAP. Entries with an asterisk denote default forwarding unless it is differently specified ........................................................................................................... 36
Figure 3.11. Translating a route to a path in RINA naming and addressing schema .................................................................................................................. 38
Figure 3.12. The mobile terminal is connected to the first AP .............................. 40
Figure 4.1. An example of a Graph......................................................................... 44
Figure 4.2 Pseudo code of generic GRASP .............................................................. 46
Figure 4.3. Pseudo code of a generic GRASP construction phase ....................... 47
Figure 4.4 Pseudo code of generic Local Search phase .......................................... 48
Figure 5.1 RINA Graph of the problem ................................................................. 51
Figure 5.3. Generic algorithm of the first phase of GRASP in RINA graph ..............54
Figure 5.4. Generic algorithm of second phase of GRASP in RINA graph ..............55
Figure 6.1. The results of the simulation for 6 demands ..................................61
Figure 6.2. The results of the simulation for 8 demands ..................................62
Figure 6.3. Simulation results: values of cost based on the different number of demands .........................................................................................................................63
Figure 6.4. Simulation results: values of cost based on different iterations (MaxIter =15) .................................................................................................................................64
Figure 6.5. Simulation results: values of cost based on different iterations (MaxIter =30) .................................................................................................................................65
Figure 6.6. Simulation Results: values of cost based on different values of $\alpha$ for 15 iterations .................................................................................................................................66
Figure 6.7. Simulation Results: values of cost based on different values of $\alpha$ for 30 iterations for 6 demands .................................................................................................................................66
Figure 6.8. Simulation Results: values of cost based on different values of $\alpha$ for 30 iterations for 8 demands .................................................................................................................................67
Figure 6.9 Simulation results: values of cost based on number of IPC processes ....68
Figure 9.10. Sample network topology (NSFNET) used in the simulation ............69
List of tables:

Table 2.1. Different packet network architecture and their naming and addressing schema ................................................................. 20
Table 3.1. Information in IDD ......................................................................................................................... 34
Table 6.1. Simulation performed and parameters used. ...................... 60
Acknowledgement:

I take this opportunity to express my profound gratitude and deep regards to my guide Professor Davide Careglio for his exemplary guidance, monitoring and constant encouragement throughout the course of this thesis. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Jordi Perello and Sergio Riccardo, for their encouragement, insightful comments, and hard questions. And moreover thank to Prof. Josep Sole Pareta who helped me a lot during my master period and introduced me to this group.

I am obliged to staff members of i2cat, Dr. Eleni Trouva and Dr. Eduard Grasa for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my master thesis.

Thanks to all my friends who helped me during this period of my life and had a hand in this project, especially thank to Mahdi Alizade who helped me a lot in the part of simulation.

Last but the least important, I owe more than thanks to my family members, especially my father, Hamidreza, and my mother, Mahnaz, who brought me to this world and have been always besides me, helping me psychologically, emotionally, physically financially to overcome all the obstacles I faced during my life, thanks to them I arrived to this level of life and I will remain always grateful toward them for what they have been doing.

This work is dedicated to my parents!!

Thank you all
Abstract

The Internet no longer meets the world’s requirements. Recognition of the Internet architecture’s inherent flaws has opened a window of opportunity for fresh thinking: granting agencies in both Europe and North America are funding large programs to lay a foundation for the next Internet.

The skills and perspective of the networking community are necessary but not sufficient to meet the needs of those who develop and deploy networked applications. The current Internet exhibits problems due to prejudice against application servers, neglect of the functional aspects of networking, poor understanding of abstraction and composition, and conflation of concerns.

By increasing the number of users and applications in internet worlds, the problems of this world are showing themselves more and more. The current internet has been facing significant challenges as these problems are growing, creating their solutions are more expensive and complicated, these difficulties will give us the perspective of new internet world to fill these insecure lacks.

Recursive Inter Networking Architecture (RINA) is a new architecture of internet which was introduced by John Day in his book [19]. This new internet architecture will give us a feasible solution to overcome these problems in a secure and simple ways rather than improving current one by its good mechanism and give us a clean state solution for the current problems.

This thesis first will have an overview about the current Internet and its architecture and will examine the challenges and problems that current internet is facing, then it will introduce the new architecture of Internet RINA and its characteristic and some views of how it is working and what advantages it has against the current old fashion one. Later on it will examine a special case of a job in RINA and will simulate it.

This simulation is based on the graph theory and use of Greedy Randomized Adaptive Search Procedure (GRASP) algorithm to find a feasible solution for this special case. GRASP will examine all the possible answers to achieve our goal.

The results of the simulation will show us what issues are important in order to achieve our objective function.
Chapter 1

Introduction

In a relatively short period of time, the Internet had an amazing impact on almost every facet of our lives. With it, we are able access to new ideas, more information, unlimited possibilities, and a whole new world of communities. It has grown and evolved to influence how we interact, how we conduct business, how we learn, and how we proceed day to day. And as much as it has changed our lives, in the process, the Internet itself has changed too.
As we know the current internet and its architecture, TCP/IP, plays an important role in everyone’s life which means that it is a vital item for living in this world and will make our life easier. As a result of its stunning success, the Internet is now being used in ways that would have been scarcely imaginable when it was designed. Not surprisingly, its architecture (as defined by accretion) is inadequate for current requirements, and often impedes efforts to satisfy them. As we know the challenges that current internet is facing are not so small that we can ignore them. In the business in the home in anyplace that you can have an access to internet you will face these problems. The challenges such as insecurity, mobility, addressing, multi homing, transporting, routing and more. So the need of a bigger world or better said secure architecture which learned lessons from previous problems and trying to avoid them and shift us one step toward a better world is obvious.

RINA [19] by its new architecture will show us how it is possible to overcome these challenges and make internet a better place to work. Results of using RINA will show us that this architecture is simpler and more reliable comparing to the current one which is complex. One of our main job is to answer a request between 2 applications and find a reliable transport according to its cost. It is really important, no one wants an expensive network!

1.1 Objective of final project

The objective of the final work is:

- A review of current internet
- study of a new architecture of internet called RINA and learning it by the challenges of the current internet
- design a routing schema for the allocation of a demand in RINA by using a best minimum cost path strategy

We used java codes in which in the appendix (b) you can find these codes. The environment of this work is Eclipse¹.

---

¹ Eclipse: Eclipse is a multi-language Integrated development environment (IDE) comprising a base workspace and an extensible plug-in system for customizing the environment. [18]
1.2. Overview of the thesis

This thesis is structured into 5 chapters:

- The second chapter will give us a background about the current internet and its architecture and also it will talk about the challenges and problems of the TCP/IP. This chapter will discuss the general characteristics and requirements.
- In the third chapter we are going to study RINA architecture (Recursive Inter Networking architecture) and more about its structure and also its characteristics. We also will study the different issues about it with comparing to the TCP/IP today and will see how RINA can overcome current internet problems.
- Chapter 4 provides some information about Graph theory and GRASP-Greedy Randomized Adaptive Search Procedure- algorithm and the steps that we have to take in this algorithm in order to get the feasible solution for our problem.
- Chapter 5 presents the solution we found about our specific problem by introducing some formulas and some objective function.
- Chapter 6 will provide us the results of our simulation and some graphical data.
- And finally the last chapter, chapter 7, is going to summarize the jobs we did and list some of the possible future works.

Moreover in the appendix we can find some knowledge about the Dijkstra algorithm which we used as the shortest path in our GRASP algorithm and our java code for the GRASP algorithm for simulating.
Chapter 2

Current Scenario of Internet

In 1961, a packet from the University of California in Los Angeles was sent from Research Projects Agency (ARPA) to the Stanford Research Institute as the world’s first packet switched network. And the hosts were connected to network through OMPs (first generation of today routers). [4]

Figure 2.1. The history of TCP/IP
In 1974, a new, more robust suite of communications protocols was proposed and experimentation began throughout the ARPANET, based upon the Transmission Control Protocol (TCP) for end-to-end network communication. The idea behind the TCP was to support the interconnection of several packets in different technologies. This design needs a way to route and transmit the packets in a reliable way between end nodes and some intelligence technologies at edges of the networks.

This chapter is structured as follow:

**Section 2.1.** Some history about the current internet.

**Section 2.2.** Absent of internet today. It gives us some information about the internet today

**Section 2.3.** Some questions about internet today.

**Section 2.4.** Defining the problem space. It will provide some information about the network architecture, some requirements and an example of it.

**Section 2.5.** Architecture of current internet.

**Section 2.6.** Some challenges internet nowadays is facing, it will present a list of the problems of internet today.

**Section 2.7.** Summary of this chapter
2.1. Some history

The design of today’s Internet technology was guided by an Internet architecture that was developed in the 1970s, under the Internet research program of the Defense Advanced Research Projects Agency (DARPA) of the US Department of Defense. Current reality and changing requirements are eating away at the viability of the original Internet architecture. Much of the coherence of the original architecture is being lost in a patchwork of technical embellishments, each intended to satisfy a particular new requirement.

The TCP/IP came next to answer the problem of growing the nodes by having some protocols from the first plans but by the time is passing it proved itself a problematic issue.

In the early 2000s, it was widely accepted that the Internet was too big to change. Network researchers could not make a proposal unless they showed how it could be adopted, incrementally, within the existing Internet. Since then, the attitude of the networking community has changed dramatically. Frustration with the incremental approach has led to a recognition that many problems cannot be solved without some “clean slate” thinking, and this recognition has stimulated a great deal of activity in the networking community world-wide. [2]

2.2 The absent of internet today

The Internet today was designed four decades ago. The scale of the Internet has grown to enormously large. With the rapid technology advancement, we now have cheap and small devices with high computing power and large storage capacity. These devices are designed to improve our daily life by monitoring our environment, collecting critical data, and executing special instructions. These devices have gradually become an essential part of our future Internet. Unprecedented amount of data are collected by these devices. How to manage and look for the desired information becomes a great challenge. At the same time, many emerging applications like service-oriented, security and real-time applications demand much better support than the current Internet can offer. [1]

The Internet has changed much in the two decades since it came into existence. It was conceived in the era of time-sharing, but has survived into the era of personal computers,
client-server and peer-to-peer computing, and the network computer. It was designed before LANs existed, but has accommodated that new network technology, as well as the more recent ATM and frame switched services. It was envisioned as supporting a range of functions from file sharing and remote login to resource sharing and collaboration, and has spawned electronic mail and more recently the World Wide Web. But most important, it started as the creation of a small band of dedicated researchers, and has grown to be a commercial success with billions of dollars of annual investment.

One should not conclude that the Internet has now finished changing. The Internet, although a network in name and geography, is a creature of the computer, not the traditional network of the telephone or television industry. It will, indeed it must, continue to change and evolve at the speed of the computer industry if it is to remain relevant. It is now changing to provide new services such as real time transport, in order to support, for example, audio and video streams.

The availability of pervasive networking (i.e., the Internet) along with powerful affordable computing and communications in portable form (i.e., laptop computers, tablets, smartphones), is making possible a new paradigm of nomadic computing and communications. This evolution will bring us new applications - Internet telephone and, slightly further out, Internet television. It is evolving to permit more sophisticated forms of pricing and cost recovery, a perhaps painful requirement in this commercial world. It is changing to accommodate yet another generation of underlying network technologies with different characteristics and requirements, e.g. broadband residential access and satellites. New modes of access and new forms of service will spawn new applications, which in turn will drive further evolution of the net itself.

The most pressing question for the future of the Internet is not how the technology will change, but how the process of change and evolution itself will be managed.
2.3. Some question about internet today

- **Will the Internet as we know it survive the next decade?**

  The Internet is facing perhaps the greatest challenge in its brief history: the distributed management of the Internet is threatened by regulation, take over and inaction (including our own!).

- **How it Works?**

  The Internet works because open standards allow every network to connect to every other network. This is what makes it possible for anyone to create content, offer services, and sell products without requiring permission from a central authority.

  It levels the playing field for everyone and it’s the reason why we have a rich diversity of applications and services that many of us enjoy today.
**Who’s in charge of the Internet?**

No one is, but everyone is!!

Unlike the telephone network, which for years in most countries, was run by a single company, the global Internet consists of tens of thousands of interconnected networks run by service providers, individual companies, universities, governments, and others.

**What’s the infrastructure of the Internet like?**

The Internet is that it’s a network of networks that needs to operate around the world as if it were one.

Like policy, the technical coordination of the Internet has common characteristics:

- Open,
- Independent,
- Run by non-profit membership organizations that work together to meet the needs everyone.

This self-regulation has been the key to the successful growth of the Internet and is flexible enough to adapt to changing future needs. [2]

### 2.4. Defining the Problem Space

#### 2.4.1. What is Network architecture?

It is important to understand what the term “architecture” means. The notion of network architecture was introduced during the Internet research phase by the research community that had developed the ARPAnet protocols. This community brought to bear on the computer communication problem the kind of abstract thinking about resources and relationships that came naturally to computer scientists who had designed hardware architectures and/or computer software systems.
Network architecture is a set of high-level design principles that guides the technical design of the network, especially the engineering of its protocols and algorithms. To flesh out this simple definition, we have examples of the constituents of the architecture and how it is applied. A network architecture must typically specify:

- Where and how state is maintained and how it is removed.
- What entities are named
- How naming, addressing, and routing functions inter-relate and how they are performed.
- How communication functions are modularized, e.g., into “layers” to form a “protocol stack”.
- How network resources are divided between flows and how end-systems react to this division, i.e., fairness and congestion control.
- Where security boundaries are drawn and how they are enforced. How management boundaries are drawn and selectively pierced.
- How differing QoS is requested and achieved.

### 2.4.2. Example: the Internet Architecture:

To further clarify the meaning of “architecture”, we can consider the Internet architecture. The important features of the original Internet architecture [5] included:

- A connectionless packet-forwarding infrastructure ("dumb network") that positioned higher-level functionality at the edge of the network for robustness ("fate-sharing").
- A single least-common-denominator data delivery service at the internetwork layer, with different end-to-end services implemented in the transport (or application) layer above. This design supports both reliable stream and (unreliable) datagram service across the same connectionless infrastructure.
- Addresses that are fixed-size numerical quantities, with a simple (net, host) hierarchy Addresses that are applied to physical network interfaces, which can therefore be overloaded for both naming a node and for routing to it.

This Internet architecture evolved during the research phase, and it has continued to evolve. The most important change during the 1974 - 1980 research period was the
separation of TCP into an inter-network (IP) layer and a transport layer (TCP), in version 4 of the protocols.

Several important features were added to the architecture during the early 1980s, especially sub netting, autonomous systems, and the domain name system (DNS). All of these changes reflecting an increasing understanding and respect for the issues of scale, imposed hierarchical design on the architecture. Sub netting was the first step towards making addresses hierarchical, and this was extended later with the addition of classless addressing (CIDR). Autonomous systems made the organization of routing hierarchical, by distinguishing EGP from IGPs. Finally, the DNS introduced hierarchical naming.

Later, IP multicasting added logical addressing and multi-destination delivery fundamental parts of the architecture.

Internet Protocol is a set of technical rules that defines how computers communicate over a network. There are currently two versions: IP version 4 (IPv4) and IP version 6 (IPv6).

IPv4 was the first version of Internet Protocol to be widely used, and accounts for most of today's Internet traffic. There are just over 4 billion IPv4 addresses. While that is a lot of IP addresses, it is not enough to last forever.

IPv6 is a newer numbering system that provides a much larger address pool than IPv4. It was deployed in 1999 and should meet the world’s IP addressing needs well into the future. The major difference between IPv4 and IPv6 is the number of IP addresses. There are 4,294,967,296 IPv4 addresses. In contrast, there are 340,282,366,920,938,463,463,374, 607,431,768,211,456 IPv6 addresses. The technical functioning of the Internet remains the same with both versions and it is likely that both versions will continue to operate simultaneously on networks well into the future. To date, most networks that use IPv6 support both IPv4 and IPv6 addresses in their networks.

Congestion control using packet loss as a congestion signal and an additive increase /multiplicative decrease algorithm at the end-systems was added in the late 1980s in response to congestion collapse events.
2.4.3. Original Requirements

Network architecture could be considered to lie in the middle between specific technical design and the overall set of technical and non-technical requirements.

For the network, therefore, a fundamental requirement of a new-arch research effort must be the choice of high-level requirements and goals. The choice of requirements may be the most critical issue determining the ultimate usefulness of the new architecture that is developed.

As an example, the following list is a brief summary of the requirements underlying the original Internet architecture. This list was ordered with the most important requirements first.
1. **Internetworking**: existing networks must be interconnected.

2. **Robustness**: Internet communication must continue despite loss of networks or routers.

3. **Heterogeneity**: The Internet architecture must accommodate a variety of networks.

4. **Distributed management**: The Internet architecture must permit distributed management of its resources.

5. **Cost**: The Internet architecture must be cost effective.

6. **Ease of Attachment**: The Internet architecture must permit host attachment with a low level of effort.

7. **Accountability**: The resources used in the internet architecture must be accountable.

---

**2.5. Architecture of current internet**

TCP/IP is a kind of open protocol, thus it enable people to fully enjoy the world sharing. However, it is because of the openness of TCP/IP protocol, the potential security risk it brings to Internet is also comprehensive and systematic. Designers did not take security into original consideration when they were developing the framework of TCP/IP protocols, because of the reliable environment of United States Department of Defense. Meanwhile, they also did not take the future large-scale application of TCP/IP protocols into account. [3]

After the Internet has been popular throughout the world, various issues arise, which includes the poor scalability and insufficient addressing space. Also, the routing table is expanding. The network is not controllable, manageable, nor accountable. The network has no perception and measurement capabilities and the quality of service can't be guaranteed [4]. According to the 2010 enterprise security report from Symantec, seventy-five percent of organizations have suffered a cyber-attack losing an average of $2 million annually [5]. This is enough to indicate that the current Internet condition is very vulnerable and cyber-attacks in nowadays have made a dramatic impact on enterprise’s operation even human’s personal life. Targeted at the current situation and issues of Internet, many countries have already begun the research for the next generation of Internet. [8]
Most people today describe the Internet architecture in terms of five layers, as shown in Figure 2.5. Each layer encompasses a collection of protocols. The Internet Protocol (IP) is the narrow waist of an hourglass, providing a common ground that allows great diversity at the top and bottom.

The five layers provide a useful vocabulary, but they are also a gross oversimplification. Real networked communication may involve multiple protocols in one of these layers, or protocols that appear to fit between the layers.

Middleware is an example of the latter. Because applications are often too difficult to build on the transport layer alone, the software community has produced middleware for message handling, remote procedure calls, distributed objects, atomic transactions, and publish/subscribe communication. This middleware fits in a layer between the application and transport layers [11], where it provides communication services to distributed applications.

![Figure 2.5. Layers of the current internet architecture](image)
2.6. Challenges that current internet is facing nowadays

What's wrong with today's TCP/IP Internet?

As we mentioned internet today is facing thousands of problems in the following paragraphs we will see some

2.6.1. Security and authentication

The Internet was designed as a network of trusted, collaborating networks. It was also designed to empower end nodes by interfering with them as little as possible. Putting these concepts together with contemporary threats and the fact that a system is only as secure as its weakest component, there is little question why security is the dominant problem.

In current internet an application can connect to another one without any permission and authentications. It seems that adding a new security is too complicated to the protocols, and is only possible if we just add this security and authentication to the protocol from the beginning.
No reliable identity authentication mechanism: Although the three-way handshake of TCP can provide synchronous confirmation and reliable communications, the support for identity authentication is very weak. Server can’t identify the identity of the log-on users, so attackers can falsify the reliable nodes’ IP addresses at will to conduct the IP address spoofing attacks.

No reliable verification means to the information integrity: Although there has the checksum’s calculation in TCP to protect the data integrity, the check to the data integrity is very weak and users can recalculate the checksum after modifying the content of packets. In addition, as TCP sequence number can be modified, attackers can add and delete any data in the original data stream.

Network security is one of the most important area of research today, a lot of research and investigation was done to improve this important issue, however the successful was partial, there are a lot of problems and threads such as DoS attacks, unwanted information (spamming), Trojans, viruses, impersonation, frauds, port-scanning, and etc. and for solving each we need to have a new technology above our system, but it is not all! There are always some bad guys who are trying to do bad things. So this is the question, can this technologies answer all the needs of their client in the secure and cheap way? The answer is no!

2.6.2. Availability

The Internet was designed to provide best effort service. Now it is so vital a part of everyday life that it should meet the availability requirement of the circuit-switched telephone network, which is to be available 99.999% of the time.

2.6.3. Mobility

By creation of mobile phones and laptops a concept of mobility will be needed more than any time before, which is also one of the challenges of internet today( which I will explain
later )and this is because of its architecture that’s why later on IP mobiles with some extra cost and overhead was added to the topology,

Because of the need for scalability in routing, Internet addresses are hierarchical. This property tends to make them location-dependent, so that a mobile device must change addresses as it moves. Providing connections to mobile devices has proved to be an enduring problem. Low performance, long delay and frequent disconnections result in degradation of the service perceived by the user.

2.6.3.1. How mobility DOESN’T work in current internet

In the current architecture when a user changes its location from an AP (access point), it will de-attachment itself and meanwhile attach to another AP. There are 2 ways of doing that:

- Horizontally: a user will change its AP to another AP but from the same technology
- Vertical: a user will change its AP to another AP but not from the same technology and can be from any other technologies

One of the main problems of TCP is that it cannot change and move vertically because by moving vertically the IP address of the AP will change and this will cause a problem in the header, because it save the same IP addresses.

Also another problems regarding to mobility in the TCP is when it changes its node from a fast to low network and vice versa because TCP doesn’t have this characteristic of updating its information so fast and easily.

2.6.3.2. Is there any solution

There is a solution to this problem in mobile network, e.g. When a mobile user changes its home router and will attach to a foreign router, the foreign router will discover the IP address of the mobile node which just attached it and will send it to the home router, so the home router will understand that the mobile node just left its place, so the foreign router will make an IP tunnel between itself and the home router and will exchange information among themselves by encapsulating the PDU (packet data unit). Seems this problems is not anymore but it will cause another problems like tunnels. IP tunnels are
hard to configure, create various security problems and represent single points of failure (if the home or the foreign agent fail, communication is lost).

2.6.4. Cost

The IANA Central IPv4 Registry was exhausted in January 2011. That was the first milestone of the IPv4 address pool depletion. Now the regional registries will start to run out and sometime later in 2011, according to the estimates by the end of the summer, there will be no more IPv4 public addresses to allocate. IPv6, by changing the address length from 32-bits to 128-bits, is bringing the solution to the inevitable depletion of the pool of unallocated IPv4 addresses. However, for most of us the migration process to IPv6 has not even started, as there is no adequate support to make the transition. Another issue is that no one is willing to pay the IPv6 upgrade cost. However, IPv6 poses a serious requirement to the router’s control planes: an IPv6 prefix consumes four times more memory in a router than an IPv4 prefix and requires more computing power for convergence of routes. This fact, compounded with the multi-homing issue and the advent of the Internet of Things billions of sensors, smart-meters and devices directly connected to the Internet may lead to routers not being able to converge on the calculations of the BGP routing tables, causing routing instabilities and ultimately an Internet far less reliable than today. [6]

2.6.5. Naming

One the problem of current internet is that there is no naming for all the entities like nodes, applications, and only there are names for POA –point of attachment- (IP addresses). This will make a problem in routing (bigger routing tables cause the routing will be in Interface level because there are only naming and addresses for them), and also in mobility and multi-homing, the result is that the network has no means to understand that two or more IP addresses of a multi-homed node belong to the same node, making multi-homing hard to realize.

The other technologies which came later didn’t follow this topology and indeed have complete naming, some technologies like XNS and CYCLADES. In 1982, Jerry Saltzer in his work “On the Naming and Binding of Network Destinations” [7] documented what XNS and CYCLADES were doing. According to him there are 4 elements that should be named
in the network, nodes, point of attachment to the network, path and applications. Which means a service can run to one or more nodes without changing its identity. This was good but also had a problem he didn’t take it into consideration that a node can reach the next hop from different path. Which would cause in routing and combinatorial explosion. The current internet did not follow his architecture and is using the one Arpanet did, TCP.

Figure 2.6. Saltzer’s point of view for a complete naming and addressing schema

Table 1 shows the components of the naming and addressing schema that the current Internet architecture has, compared to other Internetwork architectures that have existed over the years. As it becomes apparent, all the architectures except TCP/IP had already figured out what was the solution to support multi-homing and mobility, going back as far as 1973 in the case of CYCLADES. This is evidence that perhaps TCP/IP did not become the “de facto” standard for packet networking because it was the best technical solution, but due to other factors [8], a choice that causes many of the problems of today’s Internet. [9]
Table 2.1. Different packet network architecture and their naming and addressing schema

<table>
<thead>
<tr>
<th></th>
<th>ARPANET</th>
<th>TCP/IP</th>
<th>CYCLADES</th>
<th>XNS</th>
<th>DECNET</th>
<th>OSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application names</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Node addresses</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PoA address</td>
<td>Yes</td>
<td>Yes, twice (IP address and MAC address)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

One thing to take into account is choosing the right names, we should consider that the applications’ name should be location-independent (because of moving easily inside the network) and the nodes’ name should be location-dependent (to choose the smaller path and have a smaller routing table). With good choice of naming we can easily find the nodes which are belongs to a tree and which one of them is a baby node and which one is parent nodes.

Moreover, there is no services to connect the applications ‘name to the nodes. We have DSN to find a synonym name for the IP addresses. And the naming services like URI and URL will face problems also (if the socket embedded in the application changes will have problem in mobility).

2.6.6. Quality of service and the best effort

There is no consensus on how to provide different levels of service and there is no specific quality of service level. As Internet applications grow in importance to society, the lack of quality guarantees will become an increasing problem. Also, bandwidth will always be scarce on wireless links, which will soon make adjustable quality of service a necessity.

The current architecture can provide only end-to-end best effort service, which is not good because it is going to treat all packets the same way, providing a single level of service. The first solution for this problem was the Integrated Services (IntServ) with quality of
service capabilities. The main problem with IntServ is scalability, indeed it is good for the small scale not for the big systems like internet because it will ask at each flow from the router to store the state of it. After that Diffserv came which didn’t have the problem of the previous one.

Feldmann also points out serious problems with network management and problem diagnosis. From a networking viewpoint, much of the problem with the current Internet architecture can be summarized in a simple statement: The network does not know what kind of traffic it is carrying, and has no way of finding out. In addition to these major problems, it is now much too difficult to build, deploy, and maintain networked applications. This inadequacy is also recognized as related to the current Internet architecture.

2.7. Summary

In this chapter we had a review on the current internet and its architecture and a brief history about the internet today and more over the difficulties that nowadays the clients are facing when they are using this architecture.

ISP services are searching for a best way to serve their clients, but difficulties of the current internet will make their jobs complicated, so one way is to improve the current one which is more expensive and will not address all problems completely and more over will make it more complex, the other way is to create a new architecture which will provide a better way to serve the clients and do their demands.

The current internet at the first doesn’t seem so difficult and hard to work but by growing the number of users and applications and more demands from the clients we can feel the empty place of the new architecture.

Mobility, security, addressing, reliable transport are the most significant jobs that we wish from the Internet, which we can hardly find in the current one! Higher bit error rate, lower BW and longer round road trip are the results, and more TCP over hybrid in wireless is absolutely inefficient.

In the next chapter we will introduce the new architecture of internet and explore how it is working and what will differ it from the current one.
Chapter 3

RINA

Networking is only inter-process communication (IPC) said by John Day in his book “Patterns in Network Architecture: A return to fundamentals” where RINA was born.

Figure 3.1. RINA label

According to those challenges that we talked about in the previous chapter, researchers and investigators are looking to find a better solution or better way to address these problems that’s why RINA is created. Recursive Inter Networking architecture or RINA can be future architecture of internet in which learned from the current internet’s problems
and is trying to introduce a better architecture to satisfy the users and clients of the internet.

In this chapter we are going to study about:

**Section 3.1.** Mail thinks about RINA

**Section 3.2.** IPC-inter processing communication- it explained what IPC is and what characteristics has and what is in IPC.

**Section 3.3.** DIF-distributed IPC facility –

**Section 3.4** IDD-inter DIF directory

**Section 3.6.** Some characteristics about RINA at the end some comparisons to current internet

**Section 3.7.** Summary of this chapter
3.1. Main things about RINA

Networking is only based on inter processing communication (IPC), this is the main and first thing that we have to know about RINA. One of the most important thing that differ the RINA from the current internet is that there is no layer in RINA. The core element of architecture is DIF which is Distributed IPC facility which will provide IPC facility over different scope and base on what services need a client will have different policies and features. Any 2 application which wants to communicate will need this DIF services that are provided by the DIFs of lower level. So can say that RINA is a set of DIFs which will give any services to their client with some features. Which means that the first level DIF will provide facilities to the second level DIF and third and so on. You can say DIF in RINA is like a layer in TCP/IP but in the TCP/IP at each layer we have different technology and the user depend on its request should go to the desired level, but in RINA we can have all the technologies required by client in the DIFs.[16][17]

Figure 3.2 An example of RINA architecture with 3-level of DIFs.

In the picture above we can see an example of RINA in which, 3 levels of DIFs are providing the IPC services and facilities for the above levels, the first level of DIFs operates on top of the physical media and its policies are optimized to deal with the
characteristics of the physical medium. This first level of DIFs provides IPC services to the second level of DIFs, the second to the third and so on. In RINA only three types of systems exist: hosts, border routers and internal routers. As we can see there is no need for the middle boxes of today such as firewalls, NATs and others [10]

If we want to conclude this part we can say, networking is not a layered set of different functions but rather a single layer of distributed Inter Process Communication (IPC) that repeats over different scopes. Each instance of this repeating IPC layer implements the same functions / mechanisms but policies are tuned to operate over different ranges of the performance space (e.g., capacity, delay, loss).

3.1.1. Some Definitions

Before we move on we would like to explain some definition which we will use in our RINA architecture.

**Inter Processing Communication (IPC):** IPC is a main object in RINA which is in charge of any jobs that we can find in each layer of TCP/IP. In section 3.2 we can find more information regarding to the IPC. We can have as many number of IPC as we want

**Distributed IPC Facility (DIF):** DIF is like a layer in TCP/IP, it is a collection of 2 or more IPC processes that will handle the communication between 2 applications or clients. We can find more information about DIF in following sections. We can have \(n\) number of DIF depends on our network.

**Inter DIF Directory (IDD):** distributed applications, are supported by one DIF or \(n-1\) DIF which is underlying the last one and will distribute the IPC services to the layer above. IDD will create a new DIF in case of communicating the 2 applications if it is necessary.

**Distributed Application Facility (DAF):** collection of 2 or more cooperating applications processes (AP) in one or more processing system. Each processing system has an instance of IDD. Which will exchange the information using IPC, is responsible for 2 action: Discovery of the application and prepare the best DIF for executing the communication

**Distributed Application Process (DAP):** application processes which are members of the same DAF. Each DAP has naming information that has to be unique amongst peer IDDs. Except from their names, the DAPs might have synonyms internal to the DAF that facilitate operations such as searching or routing. [1]
**IDD DAPs:** belong to the same DAF (also called “peer IDDs”) are able to exchange messages for the discovery of applications.

**IDD REQUEST:** in each IDD requests we can find some information like
- Destination’s IDD DAP name
- Source’s IDD DAP name
- Requested-Application-Process-Naming-Information
- Requesting-Application-Process-Access Control Information
- QOS parameters
- Termination condition (e.g. hop count)

**Common Distributed Application Protocol (CDAP):** CDAP is RINA’s application protocol, it is used both for the application connection establishment and the application data transfer phases. CDAP is object oriented, providing 6 primitives to perform operations on objects: create/delete, read/write, start /stop. All the objects have the following attributes: class, name, instance and value.

### 3.2. IPC-Inter Processing communication

As we mentioned before each IPC can do all the jobs like routing, management, transport and authentication, etc. there are 3 main concepts in IPC which is responsible to execute all the jobs.

![Figure 3.3. Components of an IPC process](image-url)
As we can see in the picture IPC API, IPC Data Transfer and IPC Management are the main elements of the IPC.[16][17]

3.2.1 IPC API is in the client application side which will do 4 tasks

Allocate: which will allocate resources between source and destination processes (with the specific QOS and an IP port).

Send: it will send the data to destination applications, the amount of data which will be transmitted is called Service Data Unit (SDU), mean while another SDUs can combine to it or fragment from.

Receive: will receive the SDU from the Destination application.

Deallocate: will terminate the connection and will free the resources.

We can say that IPC API will actually move the data. IPC Processes are just application processes. Once they have a communication flow between them, they have to set up an application connection before being able to exchange any further information.

The application connection allows the two communicating apps to:

- Exchange naming information with its apposite, optionally authenticating it
- Agree on an application protocol and/or syntax version for the application data exchange phase
3.2.2. IPC DATA TRANSFER (it is optional)

**SDU Delimiting:** it will delimit the start and end of the SDU, and can do it by a special bit pattern.

**Error and flow control protocol (EFCP):** it consist of 2 part Data transfer protocol (DTP) which is responsible for the assembling, fragmentation, concatenation. And the DTCP (data transfer control protocol) which is responsible for the transmitting, retransmitting and flow control. The string of octets which is exchanging between DTP and DTCP is called PDU (protocol data unit) which has 2 parts, data user (is meaningless for DIF and will just send to user) and PDI (which is only used by DIF).

**Relaying Task:** it will check the destination address in PDI and will forward the PDU to the destination protocol machine (PM).

**Multiplexing Task:** Mapping of the flows of PMs belonging to a higher layer onto flows of PMs belonging to a lower layer. [10]

**SDU Protection:** it is responsible for security of SDU and will perform the mechanism like Data Corruption Protection, checksums, hop count, Time to live and encryption.
3.2.3. IPC Management

In this part we can have the security, authentication and also managing the source allocation and responding to request for allocation and de allocating resources for the requests. Moreover the RINA will be ensures that the other management task have the information they require.

3.2.4. An example of one communication in RINA

in RINA for the security issue when 2 applications want to communicate with each other they should be in the same DIF so if application A which is in DIF \( n \) wants to communicate with the application B which is in DIF level N-2, the application A should move to the same DIF that application B exists. Later I will explain more about it in details.

![Figure 3.5. IPC between two application processes in different system](image)

The steps that take place to achieve communication between the application processes S and D are the following:

- DIF A maps the source and the destination application processes names S and D to the IPC processes A1 and A3 respectively.
• Application process S using the IPC API does an allocate request to the underlying DIF A specifying the destination application process name D and the desired QoS parameters for the communication.

• The Flow Allocator in IPC application process A1 receives the request and validates it. If the request is well formed and the IPC process has enough resources to honor the request then it is accepted. The Flow Allocator in A1 creates the EFCP instances (DTP and if required DTCP) for the data transfer.

• Then, the Flow Allocator at A1 searches the local directory for the requested application process D, finds an entry that maps D to IPC process A3 and then sends a request to create a flow to A3. The request for the creation of the flow is a CDAP protocol exchange.

• The Flow Allocator at A3 receives the request to create flow and delivers the allocate request to the destination application process, D.

• Application process D submits an allocate response to the Flow Allocator in A3. The response depends on whether application S has the rights to access application D.

• The Flow Allocator in A3 creates the EFCP instances for data transfer (DTP and if required DTCP) and sends a response (CDAP) to the request for creating a flow it received to A1.

• If the response is positive, the two applications, S and D, can use the IPC API calls send and receive to send and receive data (SDUs) to/from each other.

• When the communication is over, both of them can invoke the de-allocate call to release the allocated resources. [10]

3.3. The content of DIF.

DIF is a collection of 2 or more IPC processes that will make the communication, as we said all the applications will need the IPC processes for executing their job, but where are the IPC processes located? The answer is DIF, moreover DIF will distribute the IPC facility among the applications, the size of a DIF is designable. Network management will create a new DIF on the rank N and will connect one or more DIFs (N-1). When a new member want to join an exited DIF it should be connected to the N DIF by underlying N-1 DIFs. By doing that it should know all of the members’ name of them, not their addresses. DIF of the lower layer will provide the services for the DIFs of upper level in the other words when an application wants to execute its job it will connect to the DIF of lower layer
and use its services. There are some securities issues about the DIFs which I will explain later in the following parts.

3.6 Figure. Possible schemas of DIF

As we can see in those pictures there is no size for a DIF and it can collect 2 or more IPC processes. A DIF can consists of 3 IPC processes or 2 or even more. There is no limits for creating a DIF in the size. Even one DIF can cover the whole network.

3.4. IDD: (inter-DIF Directory) ---- distributed application in DIF---- manage a namespace

When an application is trying to communicate to another one it should first knows where the requested application is located (in which level of DIF) as we know in RINA when 2 applications want to communicate with each other they should be in the same layer, BUT in here for discovering the requested application there is no need for the first application
to be in the same layer as the requested one is. (Only for discovering the request application).

IDD-Inter DIF directory- will discover where the requested application is and will handle creating the new DIF or either joining the first application in the same DIF as the requested one is! In the following paragraphs we are going to learn some definitions about this procedure.

### 3.4.1. How it works:

When an application wants to communicate with the other one first the DIFs of the source application will search itself to see if the requested one is available there or not, if the answer is negative the discovery application will be started. First all the DIFs will be discovered for looking for the requested application, if we couldn’t find any positive answer in the next step IDD will start discovering the requested application and then when it finds the requested one it will look for a good place to perform the execution. But how it works in details?!

![Diagram](image)

**Figure 3.7.** Application discovery involves discovering which layers support the requested application.

The IDD of DAP in a DAF will search all the DAPs of a DAF one by one from the source until it can find the destination application or will search until a pre-defined termination condition is met. Then it will check if the requested application is still there or not and then
it will check if it has the authorization to connect to it and make the communication. The next step is to find a good place for the communication which is to create a new DIF or either to join to an exciting one.

3.4.3 Information in IDD:

As I mentioned for discovering an application and later on for executing the job we will use IDD, so what we will need are some information about the network and the applications which we can find in IDD, information like: naming/ synonyms, Neighbor Table, Search Table, and Directory.

Each DAP in IDD has 2 table to facilitate the discovering procedure one is Neighbor and the other is Search table both of the tables have forwarding information.

The search table: “which peer IDD should I ask next for this application”. [11]

The neighbor table: “which of my nearest neighbor IDDs should I ask next to reach this destination peer IDD” [11]
The neighbor table always refer to nearest neighbors peer IDDs, but the search table will refer to the list of next peer IDDs, This table does not always point to nearest neighbors peer IDDs.

Each IDD DAP has also a directory, it contains records that have mappings of application processes information to list of DIFs information, providing the DIFs that support a requested application residing in this system. The application process information stored in the Directory is the application’s name and access control information. The DIF information maintained is the DIF names, the access control information and the Quality of Service (QoS) classes provided. [11]

3.4.4. IDD DAF as an example

In this example as we can see we have a source (IDD1) which is looking for a destination (IDD17). The orange circle that we can see presented the IDD DAPs. We assume that DAF just search its DIF and understand that none is its DIF haven’t the requested one. So it will start the discovery mechanism by send the IDD request and by looking in the forwarding table. The destination is example.google.com so depends on the information that we have on our tables we will start discovering.
Figure 3.10. An example of the IDD DAF in which the search during the first phase of the IDD function is organized according to the hierarchical organization of the application namespace. Neighbor IDDs are connected to each other (in reality a hop in the IDD graph may be several hops away).

IDD1 will see the only destination it can reach is IDD2. Then IDD2 depends on the information in the tables will continue to search the example.google.com. For example in the IDD8 we see we can reach IDD10 (.cnn) and IDD11→IFF13 (.google) and IDD12→IDD14 (.Microsoft) so it will choose IDD12 then IDD13 and so on until reaching the destination. After reaching the destination then it have to prepare a best area to do the jobs in here means DI. [11]
3.5. Some characteristics of RINA and some comparison to TCP/IP

3.5.1. Naming and addressing in RINA

In RINA, naming and addressing is on nodes or application levels not interfaces levels. Despite the TCP/IP we have naming for all entities in RINA. As we know, only 2 IPC processes can be connected to each other if and only if they are in the same DIF, because
only the members of same DIF can know the names of the others member. So if 2 process are not in the same DIF either they will change their DIF or create a new DIF, Which is done by network manager.

In the current internet there is no names for the layers there are just some IP addresses (it will make it impossible to know where the requested application is), and also there is no mechanism to show that in each layer which applications are available. there is DNS which will provides for the IP addresses an application protocol to be found if they already know the port they can connect to the requested one. And more over if there are more application protocols using the same IP addresses we will need more technologies like SIP and http to determine how to find the desired one and as we know more technologies more expensive.

### 3.5.2. Mobility and multi-homing in RINA

in RINA the addressing is on the nodes base so, for routing a node process address will connect to an interface and if that interface eventually failed the node address will map to another interface (the cost of it is small because this changing interface is local). To compare with current internet, the interface will name both nodes and the interface path, so if an interface fail, there will be a big problem. It means that naming and addressing is on nodes or application levels not interfaces levels. Attaching and de attaching from one AP to the other one in current internet is done as well as in RINA but in here we don’t have AP, we just have DIFs, it means that when a node wants to move its location, it will join another DIF and the IPC process will know how to address this routing from source to destination, when a new binding is coming it will be the last one in path from source to destination and when one wants to leave it will be the first one.

In the current internet we saw that there are major problems in mobility, one of them is adopting to different services (wired, wireless, LTE, GPRS …) while the node is moving is expensive and inefficient, that is solved by RINA, and another is in completing naming an addressing in current internet, which is also solved in RINA.

When a mobile node moves it will need a new PoA that connect it to the network, in RINA, when we have a new PoA it means that a mobile system just joins a new DIF. When a mobile is moving it will drop all its properties in the old DIF and will join another or new DIF.
To better understand how mobility works, a closer look to the naming and addressing schema used in RINA is required. Figure 3.11 shows an example of the RINA architecture with two levels of DIFs. A DIF maps the source and the destination application names to node addresses (IPC processes) through the directory function. Then, the route from a source application to a destination application is computed as a sequence of (N)-addresses (the blue line in Fig.3.11), where the next hop is a node address. Each IPC process knows how to map (N)-addresses to (N-1)-addresses for all nearest neighbors to determine the path to the next hop. Having the addressing schema of RINA in mind, multihoming could now be expressed as an IPC process having more than one (N-1) mapping and mobility as acquiring new (N-1) mappings to use and maybe losing some others.[12]

![Figure 3.11. Translating a route to a path in RINA naming and addressing schema](image)

3.5.3. An example of Hand-off in RINA:

In this section, we present a simple hand-off scenario in order to explain how transport over heterogeneous networks is handled in RINA architecture. Our scenario involves a user moving around with a mobile terminal causing hand-offs (Fig. 3.12). The mobile terminal has two different interfaces active (e.g. Wi-Fi and GPRS). An application process running on the user’s terminal has opened a connection with an application process running on a server and data transfer is taking place (e.g. the user is streaming live video). While the connection is open and packets are in transit, the user moves physically, forcing a hand-off from the one AP to the other. In our example this is vertical hand-off, as the first AP is a WLAN router (e.g. Wi-Fi) and the second AP is a router of some technology belonging to the Long Term Evolution (LTE) standard (e.g. GPRS, WiMAX).
Having both APs connected to the same router that the server is connected to is unlike to happen in reality, but we are pointing to a simplified scenario that can be easily pictured and understood by the reader. However, this choice does not spoil the generality, since the number of the possible intermediates does not affect the way mobility and hand-offs are treated in RINA.

Figure 3.12 depicts the case in which the user is in the range only of the Wi-Fi and the terminal is attached to the network through the first AP. In the lower part of figure we can see the RINA architecture for this case. We illustrate as "source" the system of the server that shares content to the mobile user and as "destination" the system of the mobile terminal. "Intermediate" is an intermediate router that connects through a wired link the server from the one side and the two APs (WLAN and LTE routers) from the other. The oval shapes are the IPC application processes residing in each system. In different colors and numbered from 1 to 7 are the DIFs formed, which are the layers of the RINA architecture. The DIFs numbered 1 - 6 are lower level DIFs (0-level or 0-DIFs), while the larger DIF, numbered 7, is an upper level DIF (1-level or 1-DIF).

We can see that the whole architecture involves only application processes that communicate with each other (IPC).
Although it is possible to have multiple layers of DIFs to achieve further configuration through policies, in our example we have chosen to describe a 2-layer RINA architecture for simplicity. The reader should keep in mind that a 0-level DIF for each media AP with a 1-level DIF for networks of a given type, i.e. technology dependent and a 2-level DIF for the technology independent service would be the norm.

The letter in each IPC process denotes its address. Every IPC process has a forwarding table with entries that map addresses of IPC processes to lower level DIFs, so that an IPC process knows where to forward an incoming PDU with a specific destination address. At the lower parts of the 0-DIFs we can see the physical links. The wireless links are denoted with dotted lines, while the wired ones are with continuous lines. As we can see in Fig.2, 0-level DIFs are sitting on top of every physical link, wired or wireless. These 0-level DIFs allow to perform different configurations through policies analogous to the characteristics of each medium. In this way the 0-level DIFs, although configuring different kind of media, provide the 1-level DIF service with certain properties (e.g. loss, delay), so that, transport at the N-DIF can operate more effectively. The 1-DIF can be seen as an \"overlay\" to which the source, the intermediate router, the Wi-Fi router, the LTE router and the destination have subscribed and it can be used by the applications of the source and the destination systems to request IPC services. The DIF numbered 4, which consists only of one IPC process, denotes that a second interface is operative in the user's mobile terminal but without a connection to an AP.

The upper DIF maps the source and destination application process names (applications running on the server and the mobile terminal respectively) to IPC process addresses (A and E). For the data transfer between the server and the mobile terminal, the route from A to E is computed as a sequence of intermediate addresses of the 1-level DIF (A, B, D, E). Using its forwarding table, each IPC process has mappings of (N)-DIF addresses to DIFs of the (N-1)-layer, in order to determine the next hop. For example, for a PDU to reach to destination IPC process E, A will forward it to DIF 1, B to DIF 2, etc. [12]

### 3.5.4. Security in RINA

In compare with the current internet RINA in its architecture has some security and authentication technique in which will make RINA more secure and will be free of cryptography issues.
3.5.4.1. Access control in RINA

In RINA if an IPC process wants to join an exciting DIF or wants to create a new DIF it should enrollment explicit. To add to a DIF, we imagine we have an IPC process which is called b and wants to make a connection with a member of this DIF which is called IPC process a. so it will send its request to (N-1) DIFs, (N-1) DIFs will determine that a is a member of N DIF and if the b can connect to it. After the registration and authentication for b, it will make an IPC channel toward a. in here b only know the name of the A and not its address, it will not know any address of IPC members of DIF, after a determines b it will assign N address to b.

In compare with the current internet, RINA in its architecture has some security and authentication technique in which will make RINA more secure and will be free of cryptography issues. How IPC managed is hidden for better security!


This chapter begins with the introduction to the RINA and an overview of what is RINA doing and how it works in details.

The problems and challenges that we faced in our current architecture of internet lead us to create a new architecture in which will work in the way that will cover all the problems of the current internet. This new architecture is totally different from the current one which is called RINA.

RINA is recursive inter-network architecture which the idea is to repeat the same layer (which is called DIF) over and over again to serve the clients demand by using IPC Process.

RINA has a major difference from the current one. In the RINA there is no layers. In only one layer the clients can perform all the jobs and requirements that are looking for. This is done by IPC process. Inter processing communication is responsible for all the requests, the requests such as routing, security, mobility, transport and etc. any application for communication to another one needs an IPC process
DIF or distributed inter processing facility is like a layer which is consists of 2 or more IPC processes. When 2 application want to communicate with each other they will need to go to the same DIF and perform their requests by IPC processes which are inside the DIF.

There is only one protocol in each DIF in which, users and applications based on their demand will use it, as we can see as a characteristic of this new architecture we don’t need too much memory to save the addressing and naming by comparing the expensive IPv6, and more over the needs of cryptography is solved, more over the cost of mobility is less than the other applications. It also will address the QOS and Scalability (based on its repeating idea).
Chapter 4.

Graph Theory & GRASP algorithm:

As this thesis used graph theory and GRASP for finding the feasible solution for our scenario so I see myself to dedicate a chapter to talk about them. We model our scenario with the graph theory and use GRASP step by step for finding our results.

In this chapter we are going to talk about:

Section 4.1 what is Graph Theory. In this section we have a review of what is graph theory. This is useful because we are going to model our problem by graph theory

Section 4.2. GRASP- Greedy Randomized Adaptive Search Procedure- some definition about GRASP and how this algorithm is working, explanation at each step what we are going to have in order to find a feasible solution

Section 4.3 summary of this chapter.
4.1. What is graph theory?

Graph theory is study of graphs, which are mathematical structure that we use to model some pairwise objects or applications. In computer science graphs are used to represent the network communication, data transition.

Graph is a set of vertices as nodes and a set of links as edges. Each graph $G = (V, E)$ is conclude the definition above. In my project I am using graph theory to define our network which is RINA. In our network each node can have more than one edges which will connect it to another nodes and for communicate inside our graph we need links between nodes.

Graph theory has abundant examples of NP-complete problems. Intuitively, a problem is called in P if there is an efficient (practical) algorithm to find a solution to it. On the other hand, a problem is called in NP, if it is first efficient to guess a solution and then efficient to check that this solution is correct. It is conjectured (and not known) that P= NP. This is one of the great problems in modern mathematics and theoretical computer science. If the guessing in NP-problems can be replaced by an efficient systematic search for a solution, then P=NP. For any one NP-complete problem, if it is in P, then necessarily P=NP [13].

A pair $G = (V, E)$ with $E \subseteq E(V)$ is called a graph (on $V$). The elements of $V$ are the vertices of $G$, and those of $E$ the edges of $G$. The vertex set of a graph $G$ is denoted by $V_G$ and its edge set by $E_G$. Therefore $G = (V_G, E_G)$.

A graph $G$ can be represented as a plane figure by drawing a line (or a curve) between the points $u$ and $v$ (representing vertices) if $e = uv$ is an edge of $G$.

The figure below is a geometric representation of the graph $G$ with $V_G = \{v_1, v_2, v_3, v_4, v_5, v_6\}$, $E_G = \{v_1v_2, v_1v_3, v_2v_3, v_2v_4, v_5v_6\}$.

![Figure 4.1. An example of a Graph](image)
4.2. Greedy Randomized Adaptive Search Procedure-GRASP

4.2.1 Introduction to GRASP

Many combinatorial problems are intrinsically difficult in the sense that to be solved they require a number of elementary steps which is exponential in the size of their input. Given their intrinsic hardness, usually such problems are approached heuristically, i.e. they are solved by applying either a heuristic algorithm or a metaheuristic schema, that find good suboptimal solutions in reasonable computational time. The most promising of such techniques include among others Simulated Annealing [Kirkpatrick, 1994], Tabu Search [Glover, 1989; Glover, 1990; Glover & Laguna, 1997], Genetic Algorithms [Goldberg, 1989], Variable Neighborhood Search [Hansen & Mladenovic, 1998], and GRASP (Greedy Randomized Adaptive Search Procedures) [Feo & Resende, 1995].

Later on in the following, we shall refer to a generic optimization problem, defined by a finite ground set $E = \{1, 2... n\}$, a set of feasible solutions $F \subseteq 2^E$, and an objective function $f: 2^E \rightarrow R$ to be minimized. The target is to find an optimal solution $\hat{S}$ in $F$, such that $f(\hat{S}) \leq f(S)$, for each $S$ in $F$.

4.2.2. What is GRASP?

GRASP is an iterative multi-start metaheuristic for solving difficult combinatorial problems. Grasp or Greedy Randomized adaptive Search Procedure is an algorithm for finding a solution for NP hard problems. This algorithm will give us a best solution, it is a multi-start or iterative process.

This algorithm will iterate as many as possible depend on network designer problem. In each iteration we have 2 phase, Construction and Local search phase. in the first phase which is Construction phase it will build a feasible solution. In the second phase which is
local search phase the neighborhood of the feasible solution from the first phase will be investigated until a local minimum solution will be found. The best over solution among the first and second phase will be kept as the result.

The pseudo-code in Figure 8.1 illustrates the main blocks of a GRASP procedure for minimization, in which Max_Iterations iterations are performed and Seed is used as the initial seed for the pseudorandom number generator.

![Procedure GRASP (MAX_ITERATIONS, SEED)](image)

*Figure 4.2 Pseudo code of generic GRASP*

Incorporation of a candidate into the solution under construction. The heuristic is adaptive because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, but not necessarily the top candidate. The list of best candidates is called the restricted candidate list (RCL). This choice technique allows for different solutions to be obtained at each GRASP iteration, but does not necessarily compromise the power of the adaptive greedy component of the method.
➢ **Construction phase:**

In figure below you can find the pseudo-code of the first phase. In the construction phase on each iteration we are going to have a feasible solution.

![Procedure GreedyRandomizedConstruction (SEED)](image)

*Figure 4.3. Pseudo code of a generic GRASP construction phase*

The key to success for a local search algorithm consists of the suitable choice of a neighborhood structure, efficient neighborhood search techniques, the fast evaluation of the cost function, and the quality of the starting solution itself, which depends on the construction phase.

The pseudo-code shows that the parameter $\alpha$ controls the amounts of greediness and randomness in the algorithm. A value $\alpha = 0$ corresponds a greedy construction procedure, while $\alpha = 1$ produces random construction. This amount of $\alpha$ will be vary from 0 to 1.

Let the set of candidate elements be formed by all elements that can be incorporated to the partial solution under construction without destroying feasibility. The selection of the next element for incorporation is determined by the evaluation of all candidate elements according to a greedy evaluation function.

This greedy function usually depends on our cost or objective function, in the way of increasing it. The evaluation of the elements by this function leads to the creation of a
restricted candidate list (RCL) formed by the best elements, i.e. those whose incorporation to the current partial solution results in the smallest incremental costs (this is the greedy aspect of the algorithm). The element to be incorporated into the partial solution is randomly selected from those in the RCL (this is the probabilistic aspect of the heuristic). Once the selected element is incorporated to the partial solution, the candidate list is updated and the incremental costs are reevaluated (this is the adaptive aspect of the heuristic).

- **Local Search Phase**

The solution which is found by the greedy randomized construction phase shouldn’t be exactly the best one. It is an optimal. That’s why we use the second step local search among the neighborhood of the feasible solution to improve the constructed solution.

```plaintext
Procedure LocalSearch (Solution)

while Solution is not locally optimal do
    Find s’∈N such that f(s’)≤f(Solution);
    Solution = s’;
endwhile
return (Solution);
end LocalSearch
```

*Figure 4.4 Pseudo code of generic Local Search phase*

In the figure above you can find the Pseudo-code of the second phase. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better solution in the neighborhood of the current solution. It terminates when no better solution is found in the neighborhood. It will have the constructed solution as an input and will start from it.
The effectiveness of a local search procedure depends on several aspects, such as the neighborhood structure, the neighborhood search technique, the fast evaluation of the cost function of the neighbors, and the starting solution itself. The construction phase plays a very important role with respect to this last aspect, building high-quality starting solutions for the local search.

A solution \(s\) is said to be locally optimal if there is no better solution in \(N(s)\). The key to success for a local search algorithm consists of the suitable choice of a neighborhood structure, efficient neighborhood search techniques, and the starting solution. [14]

4.3. Summary.

This chapter presents the basic information about the graph theory for modeling our network and our problem and also some information about the algorithm we are going to use in our scenario for finding the best and feasible solution.

Then in the next chapter we will apply them in our RINA network and later on we will analyze the results.

GRASP is an algorithm which will be applied in NP hard problems to find the feasible solution which is consists of 2 phases, construction and local search phase. In the first phase based on our greedy function we will start to find the best way to route our demands the output of this phase is not locally optimal, to find the feasible one we will apply the second phase and by checking the environment of the first solution will introduce the best one.
Chapter 5

GRASP and Graph theory in RINA:

What we did in my master thesis was trying to forward a request or demand in a RINA network from an application to the desired one, and later on calculated the minimum cost as an objective function.

We consider RINA network as a graph with some nodes and some requests to transport. In my case we have some demands with a source and destination that need to be routed in order to find the best path. This problem in RINA is equivalent to find an available DIF or create a new one. In this chapter, we will define our scenario and some works and analysis.

In this chapter we will have:

**Section 5.1.** Some definition about our problem

**Section 5.2** GRASP based re-optimization. In this section we are going to study more about our problem and performing both phases of the GRASP algorithm into our scenario more over introducing some formulas that we used in each step.

**Section 5.3.** Summary of this chapter.
5.1. Some Definition:

As I mentioned before we consider RINA network as a graph with some nodes and some links, as we said in the previous chapters each application to do its requirement needs to join a DIF and each DIF is a collection of 2 or more IPC processes to do communication. So if we define our scenario as an example we can define our RINA graph like this.

- A set of nodes or in our case IPC processes. \( V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \).
- A set of links \( E = \{(1, 2), (2, 4), (3, 4), (4, 5), (5,6), (4,7), (6,7), (7,8), (5,6), (6,8), (8,9)\} \).
- A set of demands \( D = \{(S_1, T_1), (S_2, T_2)....(S_n, T_n)\} \) which we have \( S \) as a source and \( T \) as a termination.

![RINA Graph of the problem](image)

5.2. GRASP-based re-optimization:

In order to find a path from a source to destination we have to create a DIF or join an existing DIF. Our aim is to minimize the cost of DIFs. In order to do that I proposed a methodology based on the GRASP algorithm. To do so several different issues need to
be addressed and tailored to the structural characteristics of the problem under study as a way to solve this problem I define some formula based on our algorithm.

First of all an adaptive greedy function need to be defined to perform the first phase and later to build the feasible solution by adding one element at the time. This element is adaptive in the sense that the value of it must be updated after each insertion in order to reflect the partial solution that we will get from this phase.

Second we should define a restricted candidate list to add the next element into the partial solution, in order to do so we need a restriction mechanism. Then based on the probabilistic mechanism we will choose the element from the RCL list to be routed in the network.

Then in the next phase we will improve our output based on our search strategy and finally the objective function for optimizing the problem should be defined. It can be a desired function which in our case is to minimize the cost of forwarding a demand and creating DIFs and at the end to minimize the cost of the RINA Network by minimizing the cost of creating or extending DIFs.

```
Procedure GRASP(X, MaxIter, α, N) // X as solution and N as neighborhood ,MaxIter as maximum number of iteration and α as parameter alpha
Input: current network state X
   Maximum number of grasp iterations MaxIter
   RCL parameter α
   Neighborhood function N
Output: new network state X*

1. X* <- O
2. For i = 1 to MaxIter do
3.   X' <- BuildSolution(X,)
4.   X'' <- LocalSearch(X', N)
5.   If f(X'') <= f(X') then
6.     X'' <- X'
7. End
8. End
9. Return X*
```

Figure 5.2. Generic algorithm of GRASP in RINA graph
5.2.1. Phase I. Construction phase:

Base on the algorithm, in the first phase we are going to build a solution, we have to find some neighborhood part that are more close to our objective function. In this phase as we said in the previous chapter we need to define a greedy function for each demand for our case and then based on our greedy function we are going to sort our connections. \( h_d \) is associated to the each demand. With the following formula we can find the greedy function for each demand.

\[
H_d = \sum_{i=1}^{n} X_i^d
\]

\[
X_i = \begin{cases} 
1 & \text{if } s_i = s_d \text{ or } t_i = t_d \\
0 & \text{otherwise}
\end{cases}
\]

Which is the sum of the repetition of \( S_d \) and \( T_d \) in all other demands. Based on the greedy factor of each demand we start our procedure. We will sort this greedy factor from max to min and we will start finding the best path with the demand with the maximum amount of greediness. In the other way we will sort our demand in the list \( L \) according to the greedy criterion. Then we will start building our solution adding one connection request at a time until the whole demands are forwarded and routed.

At each iteration the list \( L \) based on the parameter \( \alpha \in [0, 1] \) will be restricted to some candidates and will be stored in the RCL list contacting only the first K elements of the L. Later randomly one of these demands will be chosen randomly to be allocated in the network. The value of K is defined by \( \alpha \) in this way:

\[
K = (1 - \alpha) + \alpha \cdot L
\]
Where $L$ is the whole candidate list, and $K$ is the number of the elements in RCL list. This process will be iterated until the RCL list vector is complete. Varying $\alpha$ from 0 to 1 and its affect on $K$ will let us control the amount of the greediness and randomness in choosing the next demand for the routing. In detail we can say when $\alpha = 0, K = 1$ (greedy choose) and in the other way when $\alpha = 1, K = L$ (random choice). This randomly choose of $\alpha$ is just for being ensured that the whole list will be selected and routed inthe our network, moreover it will help us no to be stuck in the algorithm. [15]

The algorithm will work as follow:

**Procedure** BuildSolution $(X, \alpha)$

Input current network state $X$
RCL parameter $\alpha$

Output:new candidate solution $X$

1. $X \leftarrow 0$
2. Order the candidate connection requests $D = (s, d)$ in the list $\mathcal{C}L$ according to the greedy criterion $GX = \sum_{d=1}^{d_i} \{ d (S = X, D) \} + \{ d (S, D = X) \}$
3. **While** $L \neq 0$ **do**
4. $RCL \leftarrow \text{MakeRcl}(\alpha)$ //Randomized
5. $V \leftarrow \text{RandomSelect}(RCL)$
6. $X \leftarrow X \cup \{v\}$
7. Reorder candidates reflecting the choice made //Adaptive
8. **End**
9. **Return** $X$

*Figure 5.3. Generic algorithm of the first phase of GRASP in RINA graph*

After each demand which is chosen to route from the list RCL, we will remove it from the list $L$ and after computing the objective function we will try to route allocate other demands but taking into account that one demand is remove and we have to calculate again the greedy function and sorting the rest of the demands in the list $L$ and again do the rest of the steps.

### 5.2.2. Performing Local Search on the solution space:

As we said in the previous chapter the solution which is built in the first phase is not the locally optimal.so in the second phase we are trying to improve this feasible solution by
means of the local search. This phase will replace the current solution with a better one according to the iterative scheme, and it will terminate when no better solution will be found in its neighborhood. so we will use the output of the first phase as the input of the second one.

The algorithm is working as following line:

```
Procedure LocalSearch(X, N, LocalSearchType)
Input: the solution built by the construction phase X
      The neighborhood function N
      Local search type parameter LocalSearchType
Output: new network state X*
1. X* ← 0
2. For i to |X| do
3.   X ← X \ {di}
4.   X ← X ∪ {di}
5.   If f(X') ≤ f(X*) then
6.     X* ← X'
7.   End
8. End
9. End
10. Return X*
```

Figure 5.4. Generic algorithm of second phase of GRASP in RINA graph

According to this algorithm in our RINA network the second phase will remove one DIF at each step and trying to find a better one in its neighborhood base on the information we have about the other DIFs in this case by removing one DIF the weight of the other DIFs will be changed and objective function will be calculated and if it can find the better minimized shortest path again by using the Dijkstra algorithm it will update it by the current one. Again it will do it for all the DIFs until there wouldn’t be any paths left out in case DIFs.
5.2.3. The Objective function:

Our main job is to find the best shortest path (DIF) to the destination. This will be done with Dijkstra algorithm in order to find a DIF with minimum cost. We need to define a weight for our DIFs. As a main thing, cost of the DIFs depends a lot on number of IPC processes that we have on them, so we define the weight of the DIF as a formula below:

\[ C(u, v) = N + \varepsilon \]

C (u, v) is the cost of the link between u and v (in order to make a DIF from source to destination) and N is number of IPC processes needed to allocate the demands in current state of network, which in here can be from 0 to 2 based on the condition below, and then the cost of a DIF is the cost of all the links which are making this DIF from our source to our termination. For each demand we have to do the same step in order to calculate its cost depends on the DIFs which are already existed in the graph or have to be created by the network designer, we are going to have different amount of weight and at the end sum of all the Cost of all DIFs will give us the cost of our RINA network.

\[
\begin{align*}
N & = \begin{cases} 
0 & \text{if exists a path (DIF) between U and V} \\
1 & \text{if U or V (only one of them) is existed in a DIF} \\
2 & \text{else}
\end{cases}
\end{align*}
\]

As an example if we consider the network below with 2 demands, we can see how we can calculate the DIF weight, for each demand.
In this network we have 2 demands which are \( D = \{(2, 3), (6, 4)\} \). In this special case we are not employing the GRASP algorithm we just want to show how our formulas are working.

For the first demand with the source 2 and destination 3, the cost will be (according to our formulas):

\[
C(2, 3) = 2 + \epsilon, \quad \text{Weight}_{\text{DIF}} = 2 + \epsilon \quad // \text{since we don’t have any DIF between } u \text{ and } v \text{ (blue line)}
\]

For the second demand with the source 6 and termination 4, we have 2 paths one is (6, 2, 3, 4) and the other one is (6, 3, 4) the cost for the first path is

\[
C(6, 2) = 1 + \epsilon \quad \text{(since for node 2 already exist a Dif)}, \quad C(2, 3) = 0 + \epsilon \quad \text{(since for both nodes 2 and 3 already exist a Dif)}, \quad C(3, 4) = 1 + \epsilon
\]

\[
\text{Weight}_{\text{DIF}} = C(6, 2) + C(2, 3) + C(3, 4) = 2 + 3\epsilon \quad \text{(green line)}
\]

And for the next path following the same rules, the weight will be \( 2 + 2\epsilon \).

Our aim is to minimize the cost do we will choose the second path which is (6, 3, 4) to allocate the second demand.

Moreover we have an objective function as minimum of the cost of each DIF or in total the minimum cost of our network to performance the job.

\[
\text{Cost of each DIF (Objective function)} = \sum_{n \in V} IPC(n) + \epsilon \sum_{p \epsilon P} L(p)
\]
Where IPC \( n \) indicates the number of IPC processes in each node \( n \) and \( L(p) \) is the length of the path \( p \in P \) which serves demand of \( D \).

Depends on the formula we achieved above and more over the cost of the whole network will be summation of the cost of all DIFs. In this way we can calculate our objective function. And we can understand how to have the cheapest possible network.

5.3. Summary.

In this chapter, we used the graph theory and GRASP algorithm to model our RINA network and find a feasible solution for our problem, our aim is routing and finding the best DIF or DIFs (path) in our RINA network in order to minimize the cost of our network as an objective function.

To do so, we implemented the 2 phases of GRASP algorithm and trying to find the shortest path by using the Dijkstra algorithm in both phases on our RINA network and more over defined some formulas as greedy function, Objective function weight of our DIFs and how to calculate the number of IPC processes in each DIF.

By applying this algorithm and deploying those formulas we reached to some results which we will show and talk about in the 6th chapter. In the next chapter we can understand how the parameters of our network and the GRASP algorithm will change the objective function and will increase or decrease the cost, to have a network as cheap as possible.
Chapter 6

Results of the simulation:

For the simulating the RINA graph with the GRASP algorithm I use some java codes. In this chapter we can find the different values of the Cost as Objective function.

In simulation we have 2 text file as Graph.txt and Demands.txt which we can find in our simulator, in which the GRASP and RINA graph files are going to read our graph which we are simulating. In the first line of graph.txt we have to put the number of nodes and number of links and then in the following lines the whole structure of our graph.

In this chapter we are going to cover the effect of all parameters which will make a change in our objective function

Section 6.1. The parameters and the results

Section 6.2. Analyze based on the number of demands:

Section 6.3. Analyze base on the different number of iterations.

Section 6.4. Analyze based on the different value of $\alpha$.

Section 6.5. Analyze based on the number of IPC processes.

Section 6.5. Analyze based on the number of IPC processes.

Section 6.6 summery of this chapter
In order to evaluate the performance of the proposed RINA graph routing framework, I created a simple and very flexible simulation environment in java. In our simulation we can find the codes for RINA graph in which all of the links have non negative weight, the code for the GRASP algorithm, for the DIF and the Demands and different steps of the GRASP algorithm, and finally the codes of Dijkstra algorithm that we used as our routing shortest algorithm in GRASP.

6.1. The parameters and the results:

The aim of the GRASP algorithm is to find the best and feasible way for our problem, as the steps we did based on the chapter 5 formulas and GRASP algorithm and the simulation we did in java codes, we could find the solution with the minimum cost.

To mention again the problem was routing and finding the best DIF (path) in our RINA graph in order to do the client’s job which has some requests with some source and destination nodes. With the results we found after the simulation, we understand that there are some parameters which will affect the objective function.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>RINA graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Demands α MaxIter</td>
<td>varying from 1-10</td>
</tr>
<tr>
<td>Number of IPC processes as nodes</td>
<td>0.2, 0.5, 0.8</td>
</tr>
<tr>
<td></td>
<td>15, 30</td>
</tr>
<tr>
<td></td>
<td>NSFNET network</td>
</tr>
</tbody>
</table>

Table 6.1. Simulation performed and parameters used.

So we started simulating our RINA graph which I already modeled and explained in the last chapter. We run the codes to find a feasible solution the results of the simulation shows that for some demands a better DIF was found. When we tried for different graph it seems for that for the smaller graph the results of the first phase was the same as the second phase or in some cases it found another DIF but with the same weight.

In the figure 6.1 we can find an example of one result after applying the GRASP algorithm to find the feasible solution for our RINA network.
The results for our special case was:

<table>
<thead>
<tr>
<th>Demand</th>
<th>Source</th>
<th>Destination</th>
<th>DIF</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
<td>[1, 2, 4, 5, 6, 7, 8]</td>
<td>8.157</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>8</td>
<td>[2, 4, 7, 8]</td>
<td>3.137</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>8</td>
<td>[3, 4, 7, 8]</td>
<td>9.080</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
<td>[1, 2, 4, 7, 6]</td>
<td>12.143</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
<td>[1, 2, 4, 7, 8]</td>
<td>8.157</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>8</td>
<td>[2, 4, 7, 8]</td>
<td>3.137</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>8</td>
<td>[3, 4, 7, 8]</td>
<td>9.080</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
<td>[1, 2, 4, 7, 6]</td>
<td>12.143</td>
</tr>
</tbody>
</table>

Alpha is: 0.402 Objective function: 32.519

Figure 6.1. The results of the simulation for 6 demands

As we can read from the results in our network for our second demand with the source 1 and destination 8 in the phase 2 (local search) of GRASP algorithm, we found a better DIF in the way of shortest length because in this case the cost for both paths is the same. And moreover for the other demands the best path with their price is shown and at the end the value of α which is chosen randomly by the simulator and the objective function of the whole network, which is summation of cost of all the DIFs.

As we said for the GRASP algorithm the solution of the first phase is not necessary the optimal one, that’s why we tried to search in its neighborhood to find the feasible solution.

The first phase will route all the demands and find all the DIFs based on our formula and our greedy function and produce a solution. In the second phase we will remove one DIF and will try to route the same demand in the different ways to find a better DIF or a better path. And later on will do that for the other DIFs.

In the figure 6.1 and figure 6.2, we can find information about weigh of each DIF and if any better DIF is found and also the objective function of the RINA network with value of the alpha α.
6.2. Analyze based on the number of demands:

In this part we analyze the results and the objective function as cost based on the number of demands and requests in our RINA network. As we talked before the main point is to route the demands and requests in the way to minimize the cost. In the figure 6.2 below we can see how the entity of demands will affect the cost of our network.

In the formula that we declared in our GRASP algorithm, the objective function is based on the number of IPC processes and the weight of the DIF. When we want to route a demand we will look for the source and destination point, in RINA network. DAF is in responsible for the discovery procedure (which I explained in the 3rd chapter). In this case our job is not to discover the destination point, we assume that DAF did found the destination point and now is our turn to route the demands, by the GRASP.
As we can see in our chart, we understand that the value of cost will increase as the number of demands are increased. So we can conclude that in the network with more demands the cost will be increased. This is obvious that more demand in the network will make our network more expensive. So in the networks with a lot of demands, we have to change other parameters to decrease the cost.

### 6.3. Analyze base on the different number of iterations:

One of the other parameters which will affect the results is the number of iterations or MaxIter as we have in our GRASP algorithm. The procedure is that for different number of iterations we will have different results it means for each iteration we will do all the steps of the GRASP algorithm and will route all the demands and will find all the DIFS. In the figure 6.4 we can found the results of the simulation for 15 iterations (MaxIter =15)
As we can see in this chart the value of the objective function will be decreased as the number of iterations is increasing and at the end of the iterations we will have the same results as we had in the previous iterations. And at the first iterations the cost is high. In the figures below we can find different results for different graphs and demands.

In the figure 6.5 we can see decreasing of the cost for MaxIter of 30. Again here we can see that at the first, our cost was high and later by more iterations it decreased.
6.4. Analyze based on the different value of $\alpha$:

Another parameter that we analyzed during the simulation was $\alpha$. As we explained in the first phase the GRASP algorithm will select an amount for different $\alpha$ in range of 0.2, 0.5, 0.8. This can be changes during each iteration based on the amount of $\alpha$ we will have different results. Which will show the area of greediness and randomness.

In the figure 6.6 we can find the results of cost for different values of $\alpha$ for our RINA graph for MaxIter =15 and MaxIter =30
Figure 6.6. Simulation Results: values of cost based on different values of α for 15 iterations

Figure 6.7. Simulation Results: values of cost based on different values of α for 30 iterations for 6 demands
In the results of simulator we can see that for $\alpha = 0.8$ and $\alpha = 0.2$ which are randomness and greediness area we have the lowest cost or in the other way we can see that, values of the costs are decreasing, meanwhile for $\alpha = 0.5$ we have the highest cost in our network.

I think it is because in our GRASP algorithm, by greedy function we can find the most seen IPC processes, in other words these IPC processes are the point of mostly DIFs. So choosing them again for another DIFs is cheaper.

### 6.5. Analyze based on the number of IPC processes.

IPC process is another parameter of our simulations, in our special case in our RINA network. The nodes of our RINA graph are IPC processes in our case. If we can see, in the formula we defined for the objective function, number of IPC processes are one of the main parameters. In figure 6.9 we can see the results for parameter IPC processes.
By analyzing the figure above, we can understand that number of IPC process is one of the major parameter for the value of cost in which by increasing in number of IPC process, the value of cost will increase. This chart is result of simulation of the NSFNET network topology which I consider as a RINA graph. In the following figure we can find its graph, so by adding more nodes or removing some nodes our result is the same as figure 6.9.

By analyzing the figure above, we can understand that number of IPC process is one of the major parameter for the value of cost in which by increasing in number of IPC process, the value of cost will increase. This chart is result of simulation of the NSFNET network topology which I consider as a RINA graph. In the following figure we can find its graph, so by adding more nodes or removing some nodes our result is the same as figure 6.9.
So we can say that the cost of creating a new DIF or extending the same DIF mostly depends on the number of IPC processes that it includes. As far as the number of IPC process increase in the DIF the cost of it will be increased as well.
6.6. Summary:

As we did the simulation and we analyzed almost all the parameters which will make a change in our objective function as cost, the parameters such as number of IPC processes, number of demands, $\alpha$ and the number of iterations. By applying GRASP algorithm on our RINA network and by reading from the simulation results we got the following results. The value of Cost will be changed as:

- By increasing the number of demands our cost will be more expensive and will be increased.
- By increasing the number of IPC processes our cost will be increased as well.
- The cost in the greediness and randomness area will be lower than when $\alpha$ is equal to 0.5.
- By increasing the number of iteration we can see that our cost will be decreased and later it will be stable and will stay with no change. (Linear)
Chapter 7.

Conclusion and future works.

In this chapter we are going to have a conclusion about our work and also some future works.

Section 7.1. Conclusion and an overview on our thesis.

Section 7.2. Future work.
7.1. Conclusion and an overview on our thesis.

Finally we reach to our last chapter and to the conclusion, we started our journey by explaining the current internet and its history, how it works, what is its architecture and finally the problems and challenges that it is facing nowadays to serve its clients like scalability, security, mobility, routing, cost, QoS and etc. Later on we introduce the new architecture of internet-RINA- which was introduced first by John Day. In his book, then we talked about its structure and architecture and its characteristics and how it works, and some comparison to the internet today.

Moreover for modeling our scenario we presents the graph Theory and how it is and for finding the solution we used GRASP algorithm, so that we also had a review on how this algorithm works. After applying them on our RINA network and simulate the performance of how our demands can be routed in order to find the cheapest path or in RINA way DIF by defining some formulas to deploy on GRASP algorithm we got some results in the way how this cost or objective function of our problem can be changed and affected by some main parameters. Our scenario was to examine a RINA network and to route some demands in it to find the best DIFs.

After simulating I found some results and changes in our objective function. Our results show that by increasing the number of demands and the number of IPC processes we are going to have an increase in our Cost and by increasing the number of iterations we are going to have a subtle decrease in our cost and after a while we can find our cost stable. The parameter $\alpha$ which was one of the main parameter in our GRASP algorithm will also change the value of cost.

7.2. Future work.
As we talked in this thesis about RINA and a lot about DIFs. We learned that each DIF has its own quality of services, and more over it can use the services and the quality of
services that its DIF below will serve to it. In our scenario we considered that there is only one quality of service in our network, so all the DIFs can serve all the requests.

In our future work we can extend our problems, in the way that we define each DIF with some quality of services and each demand with its QoS. In the following paragraph I explained an example in order to understand it better.

As an example we imagine that in our DIF\textsubscript{3} we have some QOS such as Q\textsubscript{2}Q\textsubscript{1}, and in the DIF\textsubscript{2} which is in its lower layer we can find the QoS such as Q\textsubscript{3}Q\textsubscript{2}Q\textsubscript{1}. This DIF\textsubscript{3} can serve its clients its 2 QoS and more the QoS of the DIF\textsubscript{2}, it means that, It can serve Q\textsubscript{3}, Q\textsubscript{2} and Q\textsubscript{1} to its clients. We also studied that each request has some defined QoS which are written in its IDD request. If an application which has different QoS that the DIF which is inside it, has, It should go to another DIF. This is done by DAF. If there is already a DIF with has the QoS as our request has, will extend it and if not! It will create a new DIF according to QoS of our request.

As a work, in our future scenario we can apply this limitation to our RINA graph and based on it we can use GRASP algorithm to route our demands in order to find the best DIF and minimize cost by consideration the QoS both for our DIFs and our Demands.
Appendix (A).

Dijkstra Algorithm:

CPSC 490 Graph Theory: Shortest Path

The shortest path problem Consider the problem of finding the shortest path between nodes s and t in a graph (directed or undirected). We already know an algorithm that will solve it for unweighted graphs - BFS. Now, what if the edges have weights? Consider the dist[] array that we used in BFS to store the current shortest known distance from the source to all other vertices. BFS can be thought of as repeatedly taking the closest known vertex, u, and applying the following procedure to all of its neighbours, v.

```c
bool relax( int u, int v ) {
    if( dist[v] <= dist[u] + 1 ) return false;
    dist[v] = dist[u] + 1;
    return true;
}
```

The procedure relax() returns true if we can improve our current best known shortest path from s to v by using the edge (u, v). In that case, BFS also updates dist[v] and adds v to the back of the queue. Imagine colouring all vertices white before running BFS. Then all the vertices on the queue can be considered gray, and all the vertices that have been processed and removed from the queue are black.

We can prove that BFS works by demonstrating the following invariant: at the beginning of each iteration, dist[v] is equal to the shortest path distance from s to v for all black vertices, v. At the beginning, the invariant is true because we have no black vertices. During each iteration of BFS, we pick the closest known vertex, u, (one of them, if there are several) and execute relax (u, v) on all of its neighbors, v. Finally, we color u black (pop it from the queue). Since u was the closest vertex (to the source), any other path to u that we might discover during subsequent iterations must be longer than dist[u]. Hence, the invariant holds for u - the only new black vertex that we get during one
iteration. Eventually, when BFS terminates, dist[v] will be set to the length of the shortest path for all black (visited) vertices, v. All other vertices will have dist[] set to infinity - "unreachable". Dijkstra's algorithm

The reason why BFS does not work for weighted graphs is very simple - we can no longer guarantee that the vertex at the front of the queue is the vertex closest to s. It is certainly the closest in terms of the number of edges used to reach it, but not in terms of the sum of edge weights. But we can fix this easily. Instead of using a plain queue, we can use a priority queue in which vertices are sorted by their increasing dist[] value. Then at each iteration, we will pick the vertex, u, with smallest dist[u] value and call relax(u, v) on all of its neighbours, v. The only difference is that now we add the weight of the edge (u, v) to our distance instead of just adding 1.

```cpp
bool relax( int u, int v ) {
    int newDist = dist[u] + weight[u][v];
    if( dist[v] <= newDist ) return false;
    dist[v] = newDist;
    return true;
}
```

The proof of correctness is exactly the same as for BFS - the same loop invariant holds. However, the algorithm only works as long as we do not have edges with negative weights. Otherwise, there is no guarantee that when we pick u as the closest vertex, dist[v] for some other vertex v will not become smaller than dist[u] at some time in the future.

There are several ways to implement Dijkstra's algorithm. The main challenge is maintaining a priority queue of vertices that provides 3 operations – inserting new vertices to the queue, removing the vertex with smallest dist[], and decreasing the dist[] value of some vertex during relaxation. We can use a set to represent the queue. This way, the implementation looks remarkably similar to BFS. In the following example, assume that graph[i][j] contains the weight of the edge (i, j).
First, we define a comparator that compares vertices by their dist[] value. Note that we can't simply do "return dist[u] < dist[v];" because a set keeps only one copy of each unique element, and so using this simpler comparison would disallow vertices with the same dist[] value. Instead, we exploit the built-in lexicographic comparison for pairs.

The dijkstra() function takes a source vertex and fills in the dist[] array with shortest path distances from s. First, all distances are initialized to infinity, except for dist[s], which is set to 0. Then s is added to the queue and we proceed like in BFS: remove the first vertex, u, and scan all of its neighbours, v. Compute the new distance to v, and if it's better than our current known distance, update it. The order of the 3 lines inside the innermost 'if' statement is crucial. Note that the set q is sorted by dist[] values, so we can't simply change dist[v] to a new value - what if v is in q? This is why we first need to remove v from the set, then change dist[v] and after that add it.
m*log(n) for inserting into and updating the queue for each edge, plus n*n for running the 'for(v)' loop for each vertex u. We can avoid the quadratic cost by using an adjacency list, for a total of O((m+n)log(n)).

Another way to implement the priority queue is to scan the dist[] array every time to find the closest vertex, u.

Example 2: O(n^2) Dijkstra's

```c
int graph[128][128], n; // -1 means "no edge"
int dist[128];
bool done[128];
void dijkstra(int s) {
    for( int i = 0; i < n; i++ ) {
        dist[i] = INT_MAX;
        done[i] = false;
    }
    dist[s] = 0;
    while( true ) {
        // find the vertex with the smallest dist[] value
        int u = -1, bestDist = INT_MAX;
        for( int i = 0; i < n; i++ ) if( !done[i] && dist[i] < bestDist ) {
            u = i;
            bestDist = dist[i];
        }
        if( bestDist == INT_MAX ) break;
        // relax neighbouring edges
        for( int v = 0; v < n; v++ ) if( !done[v] && graph[u][v] != -1 ) {
            if( dist[v] > dist[u] + graph[u][v] )
                dist[v] = dist[u] + graph[u][v];
        }
        done[u] = true;
    }
}
```

We have to introduce a new array, done[]. We could also call it "black[]" because it is true for those vertices that have left the queue. First, we initialize done[] to false and dist[] to infinity. Inside the main loop, we scan the dist[] array to find the vertex, u, with minimal dist[] value that is not black yet. If we can't find one, we break from the loop.

Dijkstra's algorithm is very fast, but it suffers from its inability to deal with negative edge weights. Having negative edges in a graph may also introduce negative weight cycles.
that make us re-think the very definition of "shortest path". Fortunately, there is an algorithm that is more tolerant to having negative edges — the Bellman-Ford algorithm.

**CPSC 490 Graph Theory: Shortest Path**

**The Bellman-Ford algorithm:**

Dijkstra's algorithm is a generalization of the BFS algorithm — meaning that Dijkstra's is itself a graph search algorithm. A search algorithm can be thought of as starting at some source vertex in a graph, and "search" the graph by walking along the edges and marking the vertices. These search algorithms do not make use of the fact that we already know before-hand the entire structure of the graph. This explains why Dijkstra's algorithm cannot handle negative weights — it can only search from what we have seen so far, and does not expect new "discoveries" at some later stage would affect what we have already processed.

The Bellman-Ford algorithm is a Dynamic Programming algorithm that solves the shortest path problem. It looks at the structure of the graph, and iteratively generates a better solution from a previous one, until it reaches the best solution. Bellman-Ford can handle negative weights readily, because it uses the entire graph to improve a solution.

The idea is to start with a base case solution S0, a set containing the shortest distances from s to all vertices, using no edge at all. In the base case, d[s] = 0, and d[v] = ∞ for all other vertices v. We then proceed to relax every edge once, building the set S1. This new set is an improvement over S0, because it contains all the shortest distances using one edge — i.e. d[v] is minimal in S1 if the shortest path from s to v uses one edge. Now, we repeat this process iteratively, building S2 from S1, then S3 from S2, and so on... Each set Sk contains all the shortest distances from s using k edges — i.e. d[v] is minimal in Sk if the shortest path from s to v uses at most k edges.
Example 3: Bellman-Ford algorithm

```cpp
vector< pair<int,int> > EdgeList; // A list of directed edges (u,v)
int graph[128][128]; // Gives the weight
int n, dist[128];
void bellman­ford(int s) {
    // Initialize our solution to the BASE CASE S0
    for( int i = 0; i < n; i++ )
        dist[i] = INT_MAX;
    dist[s] = 0;
    for( int k = 0; k < n-1; k++ ) { // n-1 iterations
        // Builds a better solution Sk+1 from Sk
        for( int j = 0; j < EdgeList.size(); j++ ) { // Try for every edge
            int u = EdgeList[j].first, v = EdgeList[j].second;
                dist[v] = dist[u] + graph[u][v];
        }
    }
    // ... Now we have the best solution after n-1 iterations
}
```

The algorithm above basically implements this idea. We start with a base case S0, and repeatedly relax every edge to generate Sk+1 from Sk. Note that in the relaxation step, we don't relax an edge if dist[u] is infinity, or otherwise we may get overflow in the addition (conceptually we never want to relax such an edge anyway). Also note that the order of using the edges can affect the intermediate sets Sk, because we may first relax an edge (u,v), then relax another edge (v,w) in the same step, while choosing the reverse order of these two edges may not relax them both. However, we now show that Sn-1 is unique, and contains the shortest distance possible from s to any vertex. CPSC 490 Graph Theory: Shortest Path Proposition 4: (Correctness of Bellman-Ford) Let Sk denote the set of distances from s such that d[v] is minimal in Sk if the shortest path from s to v uses at most k edges.

Then the Bellman-Ford algorithm builds S0, S1, ..., Sn-1 iteratively. Also, Sn-1 is the best solution, and it is unique.

**Proof.** We have already establish that the Bellman-Ford algorithm generates S0, S1, ..., Sn-1 iteratively in the above paragraphs. Now, assuming that negative weight cycles reachable from the source do not exist in the graph, Sn-1 will contain the shortest possible distances from s to any other vertices. This is because any walk in the graph will go into a cycle if we use more than n-1 edges, and since negative cycles do not exist, we never
want to use these positive weight cycles as part of a shortest path. And, because Sn-1 contains the best distances, it is unique. So, the Bellman-Ford algorithm is correct, but does it always terminate? It does, as we only have two loops, one running n-1 iterations, and the other going through all edges. Hence, the algorithm always terminates, and has a run time of O( n*m ).

While the Bellman-Ford algorithm can handle negative weight edges readily, the correctness of the algorithm breaks down when negative weight cycles exist that is reachable from s. However, the nature of the algorithm allows us to detect these negative weight cycles. The idea is that, if a negative weight cycle exist, then Sn-1 will be the same as Sn, Sn+1, Sn+2, ... If we run the iteration step more than n-1 times, we will not be changing the answer. On the other hand, if a negative weight cycle exist, then one of its edges must have negative weight, and any such edge can be relaxed further even after n-1 iterations, decreasing some of the distances.

Hence, to detect negative weight cycles, we just need to run the Bellman-Ford algorithm, and when it terminates, check whether we can relax any edges. If we can, then that edge is reachable from a negative weight cycle, and the cycle is also reachable from the source.

Example 5:

Detecting negative weight cycles in a graphvector

```cpp
< pair<int,int> > EdgeList; // A list of directed edges (u,v)
int graph[128][128]; // Gives the weight
int n, dist[128];
int main() {
    // ... Set up the graph
    bellman-ford( 0 ); // Run bellman-ford on s=0
    // Check for negative weight cycles reachable from s
    for( int j = 0; j < EdgeList.size(); j++ ) { // Try for every edge
        int u = EdgeList[j].first, v = EdgeList[j].second;
        if( dist[u] < INT_MAX && dist[v] > dist[u] + graph[u][v] ) // can relax
            cout << "Negative cycle reachable from s exists." << endl;
            return 1;
    }
    cout << "No negative cycle detected, shortest distances found." << endl;
    return 0;
}
```
Bellman-Ford is slower than Dijkstra’s, but with this added functionality of handling negative weights and detecting negative cycles easily, it can be more useful in some cases. In particular, in a directed graph (one with no cycles), we can use Bellman-Ford to find the longest path from s to any vertices v, by simply changing all the positive weights to negative, and vice versa. Note that finding the longest path in a general graph is NP-hard.
Appendix (b)

Codes of simulating

Grasp:

```java
import java.util.Random;
import java.util.Vector;

public class Grasp {
    private RinaGraph rinaGraph;
    private String filename = "graph.txt", fileDemands = "demands.txt";
    private Vector<Demand> demandsVector = null;
    float alpha;
    //Number of iterations performed successfully (<= maxIter);
    //You want to run a successful iteration that has managed to find a feasible solution
    int numIterSuccessful = 0;

    //private int maxIter;

    public Grasp(int maxIter) {
        rinaGraph = RinaGraph.initGraph(filename);
        demandsVector = Demand.initDemands(fileDemands);

        //***************Phase I******************
        //Main loop Grasp
        for (int j = 0; j < maxIter; j++) {
            System.out.println("--------------Iteration Num: " + j + "------------");
            alpha = (float) Math.random();
            Vector<Demand> tmpDemands = (Vector<Demand>) demandsVector.clone();
            double weight = 0;
            while(!tmpDemands.isEmpty()){
                if (alpha != 0) {
                    //sort demands
                    tmpDemands = SortDemands.sort(tmpDemands);
                    int rcl = makeRCL (alpha, tmpDemands);
                    Vector<Demand> restrictedCandidateList = new Vector<Demand>();
                }
            }
        }
    }
}
```
for (int i = 0; i < rcl; i++) {
    Demand demand = tmpDemands.remove(0);
    restrictedCandidateList.add(demand);
}

Rina rina = new Rina(rinaGraph, restrictedCandidateList);
weight += rina.getObjectivefunction();

// Demand candidateDemand = selectedCandidate(rcl, CandidateListL);

System.out.println(" alpha is :" + alpha + " Objective function : " + weight);

//} //end of iteration

private int makeRCL (float alpha, Vector<Demand> CandidateListL) {
    //Return value that identifies the number of objects that belong to the RCL
    int rcl = 0;

    // Compute the value rcl
    // Alpha = 0 => rcl = 1 (completely greedy choice)
    // Alpha = 1 => rcl = L.size () (completely random choice)

    // Compute the rcl with its canonical formula
    float rclFloat = 1 + (CandidateListL.size() - 1) * alpha;

    // Round off the result to the nearest whole
    rcl = (int) Math.floor(rclFloat);

    // Return the number of elements that belong to the RCL
    return rcl;
}

public static void main(String[] args) {
    Grasp grasp = new Grasp(15);
}
import java.util.Vector;

public class Rina {
    private RinaGraph rinaGraph;
    private String filename = "graph.txt", fileDemands = "demands.txt";
    private Vector<Demand> demandsVector = null;
    private double Objectivefunction;

    public Rina(RinaGraph rinaGraph, Vector<Demand> demands) {
        this.rinaGraph = rinaGraph;
        demandsVector = demands;
        //rinaGraph = RinaGraph.initGraph(filename);
        //demandsVector = Demand.initDemands(fileDemands);
        //***************Phase I**************
        Vector<Demand> tmpDemands = demandsVector.clone();
        /*tmpDemands = SortDemands.sort(tmpDemands);
         System.out.println("Result of Sorting Demands:");
         for (Demand demand : tmpDemands) {
             System.out.println(demand.getDemand());
         }*/
        while(!tmpDemands.isEmpty()){
            //sort demands
            tmpDemands = SortDemands.sort(tmpDemands);
            Demand maxXDemand = selectedCandidate(tmpDemands.size(), tmpDemands);
            //tmpDemands.remove(0);
            //find diff
            Diff diff = Dijkstra.dijkstra(rinaGraph, getAllDiffs(maxXDemand), maxXDemand);
            //update diff in the demand
            for (Demand demand : demandsVector) {
                if(demand.compareTo(maxXDemand) == 0){
                    demand.setDiff(diff);
                    break;
                }
            }
        }
        //***************Phase II**************
        tmpDemands = (Vector<Demand>) demandsVector.clone();
        while(!tmpDemands.isEmpty()){ 
            //sort demands
            //tmpDemands = SortDemands.sort(tmpDemands);
            Demand maxXDemand = tmpDemands.remove(0);
            //find diff
            Diff diff = Dijkstra.dijkstra(rinaGraph, getAllDiffs(maxXDemand), maxXDemand);
            //update diff in the demand
            for (Demand demand : demandsVector) {
                if(demand.compareTo(maxXDemand) == 0){
                    demand.setDiff(diff);
                    break;
                }
            }
        }
    }
}
//find diff
Diff diff = Dijkstra.dijkstra(rinaGraph,
getAllDiffs(max XDemand), max XDemand);
//update diff in the demand
    for (Demand demand : demandsVector) {
        if(demand.compareTo(max XDemand) == 0){
            if(demand.getDiff().getWieght() >

diff.getWieght()){  
                demand.setDiff(diff);
                System.out.println("For demand " +
                demand.getId() + " ( src: " + demand.getX() + ",
+ 
                dest: "+ 
                demand.getY() + " ) a better dif is found!");

                System.out.println(" Dif phase1 is " +
                demand.getDiff().getPath() + " weight Dif Phase 1 is" +
                demand.getDiff().getWieght());
                System.out.println(" Dif Phase 2 is " +
                diff.getPath() + " weight " + diff.getWieght());
                break;
            }
        }
    }

    for (Demand demand : demandsVector) {
        System.out.println(demand.getDemand() + " diff " +
        demand.getDiff().getPath() + " weight " + demand.getDiff().getWieght());
    }
}

public double getObjectivefunction(){
    return Objectivefunction;
}

public Vector<Diff> getAllDiffs(Demand excludedDemand){
    Vector<Diff> diffVector = new Vector<Diff>();
    for (Demand demand : demandsVector) {
        if(demand.getDiff() != null &&
        demand.compareTo(excludedDemand) != 0 )
            diffVector.add(demand.getDiff());
    }
    return diffVector;
}
private Demand selectedCandidate (int rcl, Vector<Demand> input) {
    //Random number generator to select a random element in the RCL
    int element = (int) (Math.random() * rcl);
    Demand SelectedItem = input.remove(element);
    return SelectedItem;
}

/*public static void main(String[] args) {
    Rina rina = new Rina();
}*/

Candidate List:

import java.util.Random;
import java.util.SortedSet;
import java.util.TreeSet;
import java.util.Vector;

public class CandidateList {

    //Parameter alpha Grasp algorithm is used in the construction phase to determine the size of the RCL
    private float alpha;
    int k;

    private RinaGraph rinaGraph;
    private String filename = "graph.txt", fileDemands = "demands.txt";
private Vector<Demand> demandsVector = null;

public CandidateList() {
    rinaGraph = RinaGraph.initGraph(filename);
    demandsVector = Demand.initDemands(fileDemands);

    Vector<Demand> tmpDemands = (Vector<Demand>) demandsVector.clone();
    Vector<Demand> CandidateListL = SortDemands.sort(tmpDemands);

    System.out.println("Candidate List L :");
}

private int makeRCL (float alpha, Vector<Demand> CandidateListL) {
    //Return value that identifies the number of objects that belong to the RCL
    int rcl = 0;

    //Compute the value rcl
    //Alpha = 0 => rcl = 1 (completely greedy choice)
    //Alpha = 1 => rcl = L.size () (completely random choice)

    //Compute the rcl with its canonical formula
    float rclFloat = 1 + (CandidateListL.size() - 1) * alpha;

    //Round off the result to the nearest whole
    rcl = Math.round(rclFloat);

    //Return the number of elements that belong to the RCL
    return rcl;
}
}
import java.util.Vector;

public class Dif {
    Vector<Integer> path;
    private double wieght = 0;
    public Dif(Vector<Integer> path, double wieght) {
        this.path = path;
        this.wieght = wieght;
    }

    public Vector<Integer> getPath() {
        return path;
    }

    public int getN(Demand demand) {
        int n = 2;
        boolean srcExistInPath = false, destExistInPath = false;
        for (Integer node : path) {
            if (node == demand.getX())
                srcExistInPath = true;
            if (node == demand.getY())
                destExistInPath = true;
        }
        if (srcExistInPath)
            n = n - 1;
        if (destExistInPath)
            n = n - 1;
        return n;
    }

    public double getWieght() {
        return wieght;
    }
}
Bibliography:


[6]. Eleni Trouva1, Eduard Grasa1, John Day2, Ibrahim Matta2, Lubomir T. Chitkushev2, Patrick Phelan3, Miguel Ponce de Leon3 and. Steve Bunch IS THE INTERNET AN UNFINISHED DEMO? MEET RINA!


[9]. [1]. Eleni Trouva1, Eduard Grasa1, John Day2, Ibrahim Matta2, Lubomir T. Chitkushev2, Patrick Phelan3, Miguel Ponce de Leon3 and. Steve Bunch IS THE INTERNET AN UNFINISHED DEMO? MEET RINA!

[10]. Eleni Trouva1, Eduard Grasa1, John Day2, Ibrahim Matta2, Lubomir T. Chitkushev2, Patrick Phelan3, Miguel Ponce de Leon3 and. Steve Bunch IS THE INTERNET AN UNFINISHED DEMO? MEET RINA!

Transport over Heterogeneous Networks Using the RINA Architecture Eleni Trouva, Eduard Grasa1John Day, Ibrahim Matta, Lubomir T. Chitkushev2Steve Bunch3Miguel Ponce de Leon, and Patrick Phelan4 Xavier Hesselbach-Serra5

GRAPH THEORY Tero Harju Department of Mathematics University of Turku FIN-20014 Turku, Finland


A GRASP-based network re-optimization strategy for improving RWA in multi-constrained optical transport infrastructures Francesco Palmieri , Ugo Fiore a, Sergio Ricciardi

http://rina.tssg.org/ the pouzin society

http://csr.bu.edu/rina/

http://en.wikipedia.org/wiki/Eclipse_(software)

Patterns in Network Architecture: A Return to Fundamentals: A Return to Fundamentals a book by John Day