Universidade do Minho

Escola de Engenharia

Master Course in Informatics Engineering

Masters Dissertation

Easy management and user interconnection across Grid sites

Author:

Tiago Silva e Sá

Supervised by:

Prof. António Manuel Pina

October, 2012

Presentation

This document is the outcome of the last years of work I dedicated to the Masters in Informatics Engineering and also to the research projects where I was involved at the same time - namely AspectGrid, CROSS-Fire and GISELA. All these projects were centred on a common subject - Distributed Systems.

It was truly exciting to embrace such projects, expanding my knowledge about a topic that combines so many different pieces, working together to provide one of the most powerful tools in the technology world.

This document represents the last milestone on my academic path (regardless of what the future holds for me); hopefully a stepping stone for further developments in this area of computing science.

Acknowledgements

It is a pleasure to show my gratitude to those who made this 'final step' possible.

Firstly, I would like to thank Prof. António Manuel Pina, who gave me the opportunity to join him doing research a few years ago, an opportunity that has certainly changed my academic path in a very positive way. His motivation, knowledge and comprehension were important factors in my reaching this point.

I could not forget the friends and colleagues I have had since my first day at university, both from classes and research groups, who studied and worked by my side, creating a stimulating and fun environment to grow.

Also the many great professors, who shared their knowledge and experience, and all the university staff who made Braga such a splendid place to learn and live.

Lastly, and most importantly, I would like to show my love and gratitude, once again, to my family, especially my parents and my brother. Here I also include my closer, lifetime friends (you know who you are!) who play the most important role in my life. I wish I could repay everything you give me!

Abstract

Distributed computing systems are undoubtedly a powerful resource, providing functions that no other system can do. However, their inherent complexity can lead many users and institutions not to consider these systems when faced by challenges posed by the deployment and administration tasks.

The first solution for this problem is the European Grid Initiative (EGI) roll, a tool that simplifies and streamlines those tasks, by extending the tools that are currently available for cluster administration to the grid. It allows the infrastructure to be easily scaled and adopted by the institutions that are involved in grid projects such as EGI.

The second part of this work consists of a platform that enables the interconnection of computing assets from multiple sources to create a unified pool of resources. It addresses the challenge of building a global computing infrastructure by providing a communication overlay able to deal with the existence of computing facilities located behind NAT devices.

The integration of these two tools results in a solution that not only scales the infrastructure by simplifying the deployment and administration, but also enables the interconnection of those resources.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the context in which this project was developed and the motivation behind it. Two main solutions were devised as part of the same project in order to achieve a common goal - Easy management and user interconnection across Grid sites: a) the EGI roll, which allows easy and efficient configuration, installation and maintenance of Grid sites by taking advantage of the features provided by Rocks Clusters and b) the Simple Peer Messaging (SPM) tool enables the interconnection of Grid sites (where nodes are usually on a private network), providing a user-level resource architecture, as part of the User Domains project [1].

1.1 Context and motivation

The Department of Informatics (DI) got involved in European Grid projects such as EGI, by supporting various research projects and sharing its resources on the Grid. Researchers involved in these projects make use of the Grid for their experiments in diverse areas, particularly in Civil Protection (CP) related activities.

The EGI infrastructure (which evolved by former projects like the Enabling Grids for E-sciencE (EGEE), as further described in section 2.3.1) is the computing and storage foundation for a great variety of research initiatives. For the context of this work, it is important to mention the most relevant projects where the DI got involved. The EGI project has a main goal

of providing researchers with 24 hours access to a geographically distributed computing Grid infrastructure. It is also responsible for maintaining and developing the gLite middleware[2] and operating a large computing infrastructure for the benefit of a vast and diverse research community. This EGI site installed at the department is an essential testbed for the subjects being addressed, that allows testing of the management tools being developed.

In summary, the work being hereby presented has a main goal, represented on the title of this dissertation: 'Easy management and user interconnection across Grid sites'. Many products and technologies have proven their value on specific tasks (e.g., deploying a computing cluster, monitoring a server), but managing a service made of a large set of smaller components is generally challenging and time consuming.

1.1.1 UMinho projects

CYCLOPS is an EU research project in the area of Grid computing that intends to bring the Grid and the Civil Protection (CP) communities closer together, making CP communities aware of the services provided by Grid infrastructures and, at the same time, making Grid researchers aware of CP specific requirements and service enhancement needs. It outlines the importance of developing e-infrastructures and virtual organisation services to fully exploit the Grid capabilities for CP applications[1].

CROSS-FIRE is a project financed by the Portuguese government which aims to exploit the Grid infrastructure to demonstrate its potential through a CP activity application to be deployed among several independent CP related organisations[2].

The research collaboration is not limited to Europe. The Grid Initiatives for e-Science virtual communities in Europe and Latin America (GISELA) project[3], where the department is also involved, has the objective of guaranteeing the long-term sustainability of the European Union-Latin America e-Infrastructure. For this purpose, the project focuses on two inter-related

[1] UMinho's contribution to the CYCLOPS project (homepage): `https://pop.cp.di.uminho.pt/cyclops/`

[2] CROSS-Fire homepage: `https://pop.cp.di.uminho.pt/crossfire/`

[3] GISELA project: `http://www.gisela-grid.eu`

goals: 1) to implement the Latin American Grid Initiative (LGI) sustainability model rooted on the National Grid Initiatives (NGI) and 2) to provide Virtual Resource Centers with the suited e-Infrastructure and application-related services required to improve the effectiveness of their research.

1.1.2 Scaling the Grid

During the development of these projects, there was the need to find a way to automatically and efficiently deploy and manage several Grid sites[4], spread across different countries. This task raised special attention in the research team and motivated deeper work on the subject.

Despite improving the way Grid sites are maintained across different countries, the collaboration of the different institutions involved in these research projects has raised another concern. A Grid is intended to be used for running a high volume of jobs, normally with high computational and storage requirements. Running a simple job on the Grid requires many interactions between different services. This adds extra overhead to the process, increasing the time submitters have to wait for the results.

As part of this effort to improve the tools that are used to utilise and manage the Grid infrastructure, the team found a need to allow efficient management and scaling of the Grid sites. A solution was devised, using the Rocks Clusters management platform and its Rolls, culminating in a tool called EGI Roll.

The CP research projects, such as CROSS-Fire, demand a pool of computing and storage resources that can only be provided by a scalable platform such as the Grid, requiring a collaboration between many research institutions. However, these institutions don't always have the personnel resources and expertise to deploy and maintain a Grid site. With that in mind, we developed a Grid site deployment and maintenance tool - the EGI roll - that makes that task easier.

However, during the development phase of an application, the code has to be repeatedly tested for debugging. Job Management tools like the ones

[4]A Grid site, also referred as Resource Center, is the set of Grid services provided by an institution, normally administrated by its own site managers

described in Section 2.4 can be used to increase the flexibility of these computational systems by allowing the developer to easily swap the endpoint where the job will run without doing major changes in the submission process.

1.1.3 User interconnection

This work is also part of a bigger research project called User Domains[1]. User Domains gathers and combines resources from multiple sources to create a per-user, geographically distributed, heterogeneous virtualisation platform where user-provided virtual machines can be executed in user mode (without admin privileges). To address the challenge of building a global computing infrastructure it is necessary to provide a communication overlay that is able to deal with the existence of computing facilities located behind NAT devices or firewalls.

In this work we introduce SPM (section 4.1), a simple peer-to-peer messaging system based on ICE and XMPP technologies, to efficiently interconnect remote user domains[3]. Given that User Domains' performance is highly dependent on the network characteristics and there is little work regarding NAT traversal performance, we evaluated SPM performance in multiple scenarios of interest (see section 5.2).

User Domains focus on providing uniform access to resources, rich software environments and enhanced application isolation in distributed environments. Enabled by the use of virtualisation in each of the underlying resources, the system maintains independence from the administrative authorities of each infrastructure and delivers an unobtrusive user-level platform that allows the user-level execution of Virtual Machine (VM) in a batch-oriented computing cluster. A customised user-mode environment enables the user to run VMs as typical cluster jobs without imposing any persistent overhead on the execution nodes.

This work is directed towards expanding User Domains to support the interconnection of multiple infrastructures in a single user-mode computation overlay through which the network traffic, storage blocks and remote user interface are directed to the intended destinations. However, most

cluster nodes, Grid worker nodes and personal workstation that need to be connected are hidden behind NAT devices, so User Domains required an efficient method to discover and interconnect multiple entities in multiple, even private, locations. Without a central control point, entities must be able to autonomously become aware of each other existence in order to communicate. Moreover, there must be a basic mechanism for these peer entities to exchange control information that allows them to bootstrap more permanent, and eventually high performance, communication links.

1.2 Document structure

The rest of this document starts, in chapter 2, with a study about the state of the art - a description of the Distributed Systems environment where this work is inserted. It evaluates the current state of cluster and Grid technologies and some areas that still pose challenges to users and developers.

After describing that context, we dive into the implementation of the tools that were developed to tackle the previously described problems:

- Using the Cluster to Scale the Grid - chapter 3
- Connecting users across Grid sites - chapter 4

Chapter 5 takes those tools and describes how they can be applied, depicting use cases and presenting some of the achieved testing results.

Finally, chapter 6 reserves space for the conclusions of the developed solutions and describes some areas that leave room for further improvements and enhancements.

Chapter 2

State of the art

2.1 Introduction

In this chapter we present some background information regarding the environment where Distributed System area is evolving, particularly on cluster and grid computing. It is also important to understand how Network Address Translation (NAT) works, the impact it has on certain applications and how the posed challenges can be overcome by using NAT traversal techniques. We also narrow down on some other specific concepts and technologies that serve as baseline of the solution that was developed as part of this project.

2.2 Cluster

In computing world, the term 'cluster' refers to a group of independent computers combined through software and networking, which is often used to run highly compute-intensive jobs. High-performance clusters have become the computing tool of choice for a wide range of scientific disciplines. A computer cluster is a group of linked computers, working together closely thus in many respects forming a single computer, usually deployed to improve performance and/or availability, while typically being much more cost-effective than single computers of comparable speed or availability. Clusters are designed to harness the power of multiple low-cost servers to provide affordable compute power for many mission-critical applications.

Computer clusters first emerged in universities and research centres, entities that are usually characterised by tight budgets and people with computer expertise.

2.2.1 Cluster management with Rocks

Setting up a computing cluster is much more than simply hooking up additional machines to a network. Somewhere, something has to set up the machines to recognise and work with each other; something has to make sure that compute tasks are assigned to the various nodes in the best fashion. A special class of middleware, which sits between the operating systems and the applications, has evolved specifically to handle these tasks, and it goes by the name of cluster management software.

Cluster management software offers an easy-to-use interface for managing clusters and automates the process of queuing jobs, matching the requirements of a job and the resources available to the cluster, and migrating jobs across the cluster.

A Cluster manager usually is a backend graphical interface or command-line software that runs on the central node, working together with a cluster management agent that runs on each computing node. In some cases the cluster manager is mostly used to dispatch work for the cluster to perform.

The free Rocks cluster distribution[4] takes a fresh perspective on cluster installation and management to dramatically simplify version tracking, cluster management and integration. This end-to-end software stack includes the operating system, cluster-management middleware, libraries and compilers. Rocks centers around a Linux distribution based on the Red Hat Enterprise line, and includes work from many popular cluster and grid specific projects. Additionally, Rocks allows end-users to add their own software via a mechanism called Rolls[5]. Rolls are a collection of packages and configuration details that modularly plug into the base Rocks distribution. The traditional Rocks architecture, used for high-performance computing clusters, favours high-volume components that lend themselves to reliable systems by making failed hardware easy and inexpensive to replace.

Rocks FrontEnd (FE) nodes are installed with the base distribution and

any desired rolls. They serve as login and compile hosts for users. Compute nodes typically comprise the rest of the cluster and function as execution nodes.

Rocks has been successfully used to manage large clusters consisting of hundreds of nodes. Included in the standard Rocks distribution are various open-source high-performance distributed and parallel computing tools, such as Sun Grid Engine (SGE), OpenMPI and Condor. This powerful collection of advanced features is one reason why NASA, the NSA, IBM Austin Research Lab and Harvard, among other institutions, are all using Rocks for some of their most intensive applications.

On this work, we decided to use Rocks clusters as the management software, for it's a massively adopted tool and also because it is the system currently being used on the department as the cluster management tool of choice.

2.2.2 Other cluster management approaches

There are other tools that also play an important role on cluster administration.

Open Source Cluster Application Resources (OSCAR)[1] was created in 2000 by a group of organisations when it became apparent that assembling cluster software from many components was challenging and tedious. This group decided to choose 'best practices', selected among the many open-source solutions and include them in a software stack so as to make the installation, configuration and management of a modest-sized cluster easier. OSCAR is a bit different in that it installs on top of a standard installation of a supported Linux distribution. It then creates customised disk images used to provision the nodes.

Also worth mentioning is Warewulf[2], developed by the Scientific Cluster Support Program at the Lawrence Berkeley National Laboratory. This open-source toolkit is similar in many ways to Rocks and OSCAR. It works by allowing compute nodes to boot a shared image from a master node so that a systems administrator only needs to support the master node and the

[1] OSCAR project homepage: `http://svn.oscar.openclustergroup.org/trac/oscar`
[2] Warewulf homepage: `http://warewulf.lbl.gov/trac`

shared image for the rest of the system. Note that starting with Warewulf 3, cluster-provisioning features have been replaced by a project called Perceus.

2.3 Grid

Grid computing has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation.

The Grid can be thought of as a distributed system with non-interactive workloads that involve a large number of files. What distinguishes grid computing from conventional high performance computing systems, such as cluster computing, is that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed. Although a grid can be dedicated to a specialised application, it is more common that a single grid will be used for a variety of different purposes. Grids are often constructed with the aid of general-purpose grid software libraries, or middleware.

The distributed computing grid was originally conceived in 1999 to analyse the experimental data produced by the Large Hadron Collider (LHC)[3] at CERN – the European particle physics laboratory located on the Swiss/French border[4].

2.3.1 EGI project

The European Grid Initiative (EGI) project, launched in May 2010, is a collaboration between National Grid Initiatives (NGIs) and other European international research organisations. The main goal of this pan-European infrastructure is to provide scientists a powerful set of computational tools, where they can take their research activities.

The European DataGrid Project[5], which started in January 2001, led the research and development of Grid technologies. It established the organisational structure, gathered and analysed requirements, developed mid-

[3] About the LHC project: `http://public.web.cern.ch/public/en/LHC/LHC-en.html`

[4] More about CERN: `http://public.web.cern.ch/public/`

[5] DataGrid project homepage: `http://eu-datagrid.web.cern.ch/eu-datagrid/`

dleware (the software that links hardware resources), and provided training to its users. The project proved the Grid's successful application in various research fields such as high energy physics, Earth observation and bioinformatics.

Upon completion in March 2004, a new project called Enabling Grids for E-sciencE (EGEE)[6] took over the Grid's further development in what would result in three successive two-year phases. EGEE provided researchers with access to computing resources on demand, from anywhere in the world and at any time of the day. Ease of access and the ability to analyse a larger amount of data within a shorter timescale than before attracted participation from a wider range of scientific disciplines.

By April 2010, when the last EGEE project phase was completed, a new project took over - the EGI, a sustainable project that links more than 18,000 researchers (and counting) to the distributed computing electronic resources (computing and storage) they need for their work.

In other words, EGI provides a production quality Grid infrastructure distributed among most European countries (including Portugal). EGI is open to new sites that want to join the infrastructure and offer their resources, sharing more power to the community. However, since EGI is a production quality infrastructure, several administrative and technical procedures exist, to ensure that a new site that is being integrated in the infrastructure will be able to meet certain quality criteria. So, before entering into production, a site must pass a set of specific registration and certification steps.

Each NGI groups a set of institutions among a geographical region, commonly formed by one or more countries. Portugal and Spain have joined their efforts in a single NGI called Ibergrid. Each NGI integrates a Regional Operation Centre (ROC), responsible for the sites operating in its region. Notably, a ROC is committed to provide know-how, advice and support to the sites for the resolution of possible installation and operational issues. It ensures that the sites get adequate support during deployment and operations and that the operational procedures are enforced within the whole

[6]EGEE project homepage: `http://www.eu-egee.org`

NGI.

2.3.2 Grid Middleware

A Grid middleware is a software tool that provides an abstraction layer between physical resources, even when provided by different vendors and operated by different organisations, and applications or users. It facilitates interoperability among Grid services and provides a simplified user interface that ultimately improves utilisation and adoption. In order to address the end-user requirements, the services need to work together in a coordinated manner, although individual services can still be deployed and used independently.

The gLite middleware[7] is an example of such a tool, currently installed in hundreds of sites participating in the EGI. gLite provides the necessary tools and configurations that allows a seamless integration of heterogeneous clusters in the EGI infrastructure. The middleware includes services and components addressing four technical areas:

Compute: processing and management of user requests concerning the execution of a computational task. Covers the interaction with Local Resource Management System (LRMS), the provision of a common interface to the computational resources of a site (the so-called Computing Element) and the availability of high-level meta-scheduling, workflow execution and task tracking functionality.

Data: storage management, data access and data replication. Multiple storage solutions exist, addressing different types of resources – disk, tape or a combination of the two – all exporting the same Storage Resource Manager (SRM) interface (the so-called Storage Element). Data access libraries are available for storage systems not offering a POSIX interface towards the computational resources. Data and metadata catalogues track the location of copies of data chunks in multiple places.

Security: enable and enforce the Grid security model, allowing the safe

[7] gLite website: `http://glite.cern.ch`

sharing of resources on a large scale. They cover identity management, Virtual Organization membership management, authentication, delegation and renewal of credentials, and authorisation.

Infrastructure: information and management functionality to deployed Grid services. They include the Information System and Service Registry, which provide a view of the services available on the Grid together with their characteristics; the messaging infrastructure, that allows to collect and distribute messages generated by Grid services or user tasks; the service monitoring and management providers that allow the retrieval of Grid services status information and service state management; the Logging and Bookkeeping services that allows collecting, aggregating and archiving job execution information; the accounting functionality to collect, distribute and publish information concerning the usage of resources This area also deals with internal infrastructure components, such as service containers that are required for middleware services.

In practical terms, it consists in a set of packages that define the different roles of computers in a cluster wishing to become part of the EGI grid[6]. Among these roles (called elements in the gLite's terminology), one can commonly find in a Grid site:

Computing Element (CE) responsible for the management of the submitted jobs and for the local resource management system (LRMS)

Storage Element (SE) provides uniform access to data storage resources. The Storage Element (SE) may control from simple disk servers to large disk arrays. Most EGI sites provide at least one SE which can support different data access protocols and interfaces

Accounting Processor for Event Logs (APEL) an accounting tool that collects accounting data from participating sites

Berkeley Database Information Index (BDII) responsible for the information system. It collects information from the site elements (such as number of CPUs, available storage, available services) and makes it available to be queried from the outside

Worker Node (WN) computing units with the processing power to execute the jobs

User Interface (UI) the access point to the grid. From a UI, a user can be authenticated and authorised to use the EGI resources, and can access the functionalities offered by the Information, Workload and Data management systems. It provides CLI tools to perform Grid operations.

In the EGEE era, the middleware development and the infrastructure operation was handled inside the same European project - as the gLite middleware was developed by the EGEE project. With the end of EGEE, different projects have emerged raising a different paradigm with respect to what was done in the past.

Under the new context, the operation of the infrastructure and the development of the middleware are responsibilities of different European projects. EGI and NGIs, and the communities using the grid infrastructure, are now assuming a client role, installing products from Technology Providers such as European Middleware Initiative (EMI).

The products developed by these external Technology Providers (i.e. EMI) have to fulfil a given set of quality criteria defined by EGI, and only the products which are properly verified and checked against the EGI quality criteria can integrate a Unified Middleware Distribution (UMD) release. The proposed approach of handling middleware maintenance, integration, testing, and deployment within the EGI infrastructure. UMD defines components, processes, involved parties, and so on, in order to guarantee the infrastructure to get reliable middleware, in terms of both functionality and quality. Under the current approach, a product can be release in EMI, for example, but never reach to integrate a UMD release.

The gLite middleware, originally produced by the EGEE project, is currently being developed by the EMI project, a Technology Provider. The EMI project aims to deliver a consolidated set of middleware products based on the four major middleware providers in Europe - ARC, dCache, gLite and UNICORE. The products, managed in the past by these separate providers, are now developed, built and tested in collaboration, for deployment in EGI

(as part of the UMD), and other distributed computing infrastructures.

The migration from former packages like gLite to the UMD is being enforced at the time of writing. Different NGIs devised an action plan that has to be applied by all the participating Grid sites, in order to ensure a seamless transition to the new platform.

2.4 Job Management

Scientific users are driven by a requirement to get computing results quickly and with minimal effort. These users have a wide variety of computing resources at their disposal, namely workstations, batch systems and computing Grids.

Each of these resources has a unique user interface and users have the burden of continually reconfiguring their applications to take advantage of the diverse systems.

Ganga Ganga[7] is a modular, user-friendly job management tool for scientific computing that supports an extensible suite of execution backends, allowing users to easily run jobs on their local workstation, on a number of batch systems and many Grid platforms. This situation is shown in Listings 2.1. The first line shows the command that lists the installed backends, depending on the experiment specific plugins that were loaded. The last three lines show an example about how to specify the desired backend, in this case: Local (run jobs in the background on local host), PBS (submit jobs to Portable Batch System) and LCG (submit jobs to the EGI/LCG Grid using gLite/EDG middleware).

Listing 2.1: Ganga usage example

```
In [1]: plugins("backends")
Out[1]: ['LSF', 'Remote', 'PBS', 'Condor', 'SGE', 'Batch', 'LCG', '
    Local', 'Interactive']

In [2]: job=Job(application=Executable(exe='/bin/hostname'))
In [3]: job.backend='Local' #job.backend='PBS' #job.backend='LCG'
In [4]: job.submit()
```

The typical workflow of Ganga is illustrated in figure 2.1. Users can easily plug new modules to suit their needs. They can develop and debug on the local workstation, then test on a batch system and finally run a full analysis on one of the many Grids.

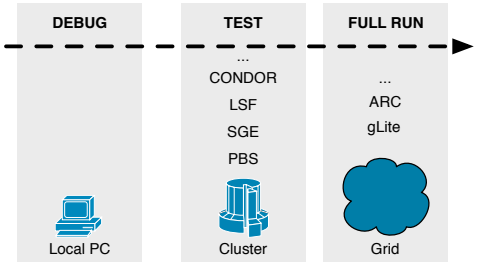


Figure 2.1: Ganga Workflow

2.5 Advantages and challenges of NAT

The Internet's original uniform address architecture, in which every node had a globally unique IP address and could communicate directly with every other node, has been replaced with a new real Internet address architecture, consisting of a global address space 'separated' from many private address ones.

Minor portions of the IPv4 address space have been allocated or assigned directly by the Internet Assigned Numbers Authority (IANA)[8] for global or other specialised purposes, namely for private addressing[8]. Routers are required to interconnect different networks, regardless of whether the IP address range is public or private. However, if the address range is private, packets cannot be directly routed across the public Internet.

In the present, most networks use NAT to address this 'issue' and it is estimated that more than 73% of the hosts are behind NAT devices[9]. This was not a problem in the past, when most communication was based on the client/server paradigm. However, the Internet has evolved and, nowadays, the largest portion of the traffic comes from peer-to-peer (p2p) applications. Applications such as p2p file sharing, VoIP services and the online services

[8]Internet Assigned Numbers Authority (IANA): `http://www.iana.org/`

of current generation require clients to be servers as well, thereby posing a problem for users behind NAT devices, as incoming requests cannot be easily correlated to the proper internal host, potentially requiring substitution or special traversal techniques for NAT traversal.

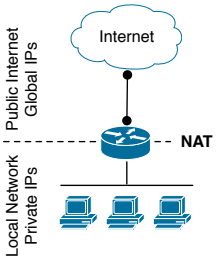
Basically, NAT is the process of transparently modifying IP address information in IP packet headers, while in transit across a routing device (see figure 2.2a). Traditional NAT[10] has two variations, namely, Basic Network Address Translation and Network Address Port Translator. The latter, Network Address Port Translator (NAPT)[11] is by far the most commonly deployed NAT device as it allows multiple private hosts to share a single public IP address simultaneously.

The main advantages of NAT are that private IP addresses can be reused and many hosts on a single LAN can share globally unique IP addresses, improving the scalability of the address space and helping to overcome the shortage of unique public addresses in IPv4. NAT operates transparently and helps shield users of a private network against access from the public domain, hiding private IP addresses from public networks. NAT operates much like an Access Control List (ACL), not allowing outside users to access internal devices. This control of both inbound and outbound traffic can be seen as an advantage, providing a level of security.

On the other hand, one disadvantage is that additional configuration is required to allow access from legitimate, external users or p2p applications. Also, it may impact on some applications that have IP addresses in their message payload, because these IP addresses must also be translated. Figure 2.2b summarises the main advantages and disadvantages of NAT.

2.6 Interconnecting hosts behind NAT

Many techniques exist for NAT traversal, relying on different methods and tools, that make these connections to hidden hosts possible. However, the success rate of different techniques depend on the NAT device being used and their the implementation can be complex. Most of the NAT traversal solutions are either part of proprietary applications or nested in the application code. Unfortunately, there is not a single technique that works



(a) Concept

Advantages of NAT	Disadvantages of NAT
• Public IP address sharing • Transparent to end users • Improves security and scalability	• Incompatible with certain applications • Hinders legitimate remote access • Increased routing processing

(b) Advantages and disadvantages of NAT

Figure 2.2: General NAT concept, advantages and disadvantages

with all existing NATs, because NAT behaviour is not standardised. There are studies determining average success rates using different methods and tools[12, 13].

Although the rules of the game are changing, with the actual migration from IPv4 to IPv6 [14, 15], eventually reducing the need for NATs connecting private hosts to the Internet, the demand for NAT will be increasing in the near future because NAT itself provides the easiest way to achieve interoperability between both versions of the IP [11]. Additionally, the migration is today an hot topic where opinions diverge. Some say that the total transition will never occur.

Internet Engineering Task Force (IETF) published the informational Request For Comment (RFC) 5128 in 2008, documenting the various AT traversal methods known to be in use by that time[16]. It covers NAT traversal approaches, used by both TCP-based and UDP-based applications. An effort was made to keep this text as coherent as possible with the terminology and conventions used on that document.

Several non-proprietary solutions have been proposed to allow network protocols to operate through NATs focused particularly on the UDP traffic required to address the widely used VoIP and file sharing peer-to-peer applications. IETF has even published RFCs 3489, 5128, 5245, 5389 and 5766 regarding NAT traversal approaches, including support for TCP applications.

The success rates using different methods and tools in multiple network configurations were presented in [8, 9], where it can be verified that com-

plexity of the network topologies largely determines each techniques' degree of success. So, there is no universal NAT traversal solution, but several techniques that can be used together depending on the network configuration.

2.6.1 NAT reconfiguration

NAT traversal strategies such as Network Address Translation-Port Mapping Protocol (NAT-PMP), Universal Plug and Play (UPnP) work with a reconfiguration of the router/NAT device, involving explicit signalling between applications and NAT devices. These methods are not always an alternative because, under certain network administration environments, a common privileged user may be prohibited to change the NAT configuration due to security concerns. Moreover, the NAT device may not even support these functionalities.

For these reasons, NAT reconfiguration strategies cannot be completely trusted as a failsafe solution.

2.6.2 Relaying

Relaying is the most reliable method for implementing p2p communication across NAT devices, but the least efficient too. This method breaks a peer-to-peer communication into two standard client/server communications, through a relaying server with a publicly addressable IP address (see figure 2.3). This method will always work, as long as both clients are able to connect to the server. The big disadvantage is that all the traffic is routed through the relay server, consuming the server's processing power and network bandwidth, and increasing the latency of the communication. However, this strategy can be useful as a fall-back when there is not a more efficient alternative and if maximum robustness is required. The Traversal Using Relays around NAT (TURN) protocol [17] defines a method of implementing application agnostic, session-oriented, relaying in a relatively secure fashion.

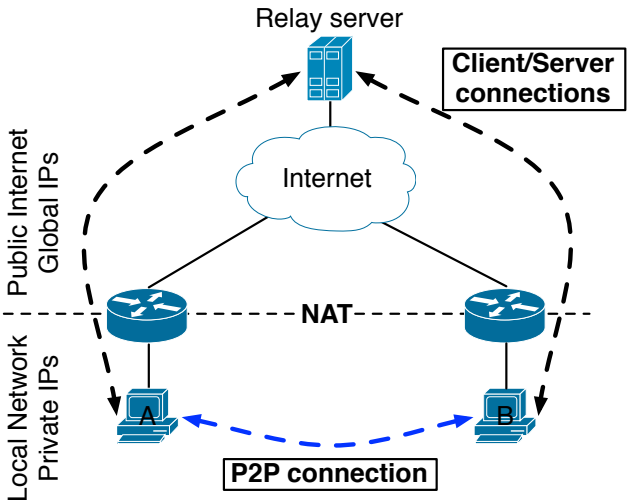


Figure 2.3: Relaying example. The peer-to-peer connection is split into two client/server connections.

2.6.3 Autonomous NAT traversal

This novel technique presented in [18] provides an autonomous method for establishing connections to peers behind NAT without reliance on a third party server. Using third parties increases the complexity of the software and potentially introduces new vulnerabilities. Technically, this method uses fake ICMP messages to initially contact a peer behind a NAT.

This strategy works best if only one of the hosts is behind a NAT device. In that case, it is assumed that the client has somehow learned the current external IP address of the server's NAT (i.e. from a previous connection between the two hosts or by exchanging that information using other out-of-band methods). Further complications arise if both hosts are behind NAT, depending on its type (RFC 3489 specifies four NAT variations[19]).

2.6.4 Multiple techniques bundle - ICE

Traditional NAT traversal methods require the help of a third party server (also known as *rendezvous* server) with a publicly addressable IP address for facilitating the connection[20, 17]. Some methods use the server only when establishing the connection, while others are based on relaying all

data through it, which adds bandwidth costs and increases latency, detrimental to real-time voice and video communications. In a few words, the basic approach in most of these cases is that the server in the private network behind the NAT is notified by the third party that the client would like to establish a connection. In order for this method to succeed, the server must maintain a connection to a third party known service, the client must also be able to locate that broker, which must act according to a specific protocol.

A method known as *hole punching* is one of the most effective for establishing peer-to-peer communication protocols between hosts on different private networks behind NAT. This technique was firstly designed for UDP-based applications, but can also be applied for TCP[12].

Interactive Connectivity Establishment (ICE)

As described above, numerous solutions exist allowing protocols to operate through NAT. Unfortunately, these techniques all have pros and cons, which make each one optimal in some network topologies, but a poor choice in others. By using them, administrators are making assumptions about the networks' topologies in which their solutions will be deployed and introducing greater complexity into the system. It is necessary to find a single solution that is flexible enough to work well in all situations.

RFC 5245, published by IETF in April 2010, defines Interactive Connectivity Establishment (ICE) as a technique for NAT traversal for UDP-based media streams (though ICE can be extended to handle other transport protocols, such as TCP) established by the offer/answer model[21]. ICE works by including multiple IP addresses and ports in offers and answers, which are then tested for connectivity by peer-to-peer connectivity checks. The IP addresses and ports included in the connection offer and the connectivity checks itself are performed using the revised STUN specification [20], now renamed to Session Traversal Utilities for NAT. This name and specification revision reflects STUN's new role as a tool that is used with other NAT traversal techniques (namely ICE) rather than a standalone NAT traversal solution, as the original STUN specification was.

In a typical ICE deployment, we have two endpoints (known as agents) that want to communicate. They are able to communicate indirectly via some signalling protocol (see section 2.7), by which they can perform an exchange of messages. At the beginning of the ICE process, the agents have no information about the network topology. They might not even be behind a NAT. Using ICE, agents can discover enough information about their topology to potentially find one or more paths by which they can communicate.

2.6.5 Other real world employed techniques

NAT traversal is currently a hot topic, due to the change on the communications paradigm (moving from client/server to peer-to-peer). Although there is a big effort to standardise the protocols and techniques, there is not a single, infallible solution to solve this problem with all existing NATs implementations. Generally, the solution is to try different strategies until the working one is found.

This may sound strange when we have plenty of successful p2p applications such as VoIP (e.g. Skype), online games, p2p file sharing (e.g. BitTorrent), p2p streaming (e.g. PPLive) that have become tremendously popular.

So, how do they overcome the NAT traversal problem? The answer is not always easy to find because these solutions are either part of closed, proprietary applications, or nested in the application code itself. Some of these popular p2p applications are described bellow.

Skype[9], is a good example of a popular Voice over IP (VoIP) p2p application, claimed to work almost seamlessly across NATs and firewalls, with good voice and video quality. Although users may find it very similar to other clients such as MSN messenger[10] and Google Talk[11], the underlying protocols and techniques it employs are quite different[22].

Skype could not work properly without efficient NAT and firewall traversal functions. According to the study by Baset and Schulzrinne[22], each Skype client uses a variant of Session Traversal Utilities for NAT (STUN)[20]

[9]Skype homepage: `http://www.skype.com`

[10]MSN messenger: `http://messenger.msn.com`

[11]Google Talk: `http://www.google.com/talk`

and TURN[17] protocols to determine the type of NAT and firewall it is behind. Also, these authors believe that 'it is by the random selection of sender and listener ports, the use of TCP as voice streaming protocol, and the peer-to-peer nature of the Skype network, that not only a Skype client traverses NATs and firewalls but it does so without any explicit NAT or firewall traversal server'. Ahmed and Shaon also have an interesting publication on this topic[23], evaluating popular VoIP services (mainly Skype, GTalk and Gizmo).

Spotify[12] is a popular music streaming service with more than ten million users. It uses a hybrid communications architecture, composed by both client-server streaming and a client p2p network overlay in order to offload the central servers. This p2p network requires a connection between clients, generally behind NAT devices. Goldmann and Kreitz recently conduced a study where, among other measurements, they describe the NAT traversal strategy employed in that application[24]. According to the authors, the Spotify client application uses two NAT traversal methods: a) NAT reconfiguration methods such as UPnP and NAT-PMP to open a port for incoming connections, as described in section 2.6.1 and b) various hole punching techniques, described in section 2.6.4.

Our conclusion is that these valuable networking features, implemented by every big company dealing with p2p applications, are well protected and patented. So, the alternative for a researcher with a quite shorter budget is to stick with the public standards, specifications and frameworks.

2.7 Control Information Exchange

This section describes the solution for indirectly exchanging control information to interconnect different devices. As described in section 2.6, while client/server model has a stable behaviour across the internet, the increasingly important p2p model sometimes raises significant problems on the presence of NAT devices.

We previously discussed different methods and solutions to overcome those problems. Among those, the most flexible and resilient solution is

[12]Spotify homepage: `http://www.spotify.com`

the use of ICE (further described in section2.6.4). However, to be able to establish a direct p2p connection between two hosts using this technique, we must firstly be able to exchange calling/control information between them.

This task falls on a different field of communication protocols, where many alternatives exist. Session Initiation Protocol (SIP) and XMPP are just two of the various possibilities.

XMPP Extensible Messaging and Presence Protocol (XMPP) is a widely deployed open technology for real-time interaction, using Extended Markup Language (XML) as the base format exchanging information. Although common users don't notice it, XMPP is under the hood of massive communication services used worldwide like Google Talk. Another important feature is its security. It provides built-in support for encryption and strong authentication. The fact that XMPP technologies are deployed in a decentralised client-server architecture with an unlimited number of servers also improve its security and scalability, since there is no single point of failure. Any person or organisation can run their own XMPP server and connect it to the rest of the network using the Internet infrastructure.

XMPP is, at its core, a technology for rapidly delivering XML between clients, being used for a wide range of applications beyond instant messaging, including gaming, social networking, VoIP, real-time collaboration and custom applications. Also, it is a standard. Its core aspects have undergone rigorous public review within the IETF, resulting in strong technologies that can be freely implemented under any licensing terms.

As you can further read on [25], these and other aspects make more and more software developers and service providers adopt this technology on their projects to solve real-world problems.

Among core services such as channel encryption, authentication, contact lists messaging and notifications, XMPP enables peer-to-peer media sessions. This service, defined in XEP-0166 allows to negotiate and manage a media session with another entity, for the purpose of voice chat, video chat, file transfer and other real-time interactions.

XMPP technologies use a decentralised client-server architecture, as depicted in figure 2.4, where we can have several clients communicating, using

a network of servers.

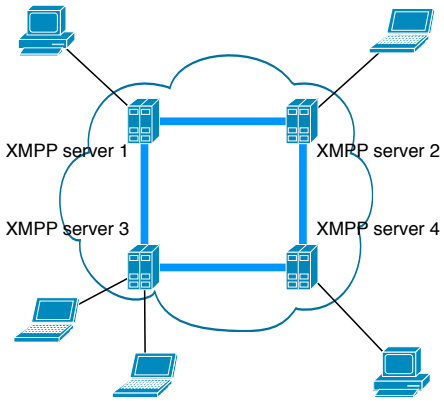


Figure 2.4: XMPP client-server, decentralised architecture

Like email, when a XMPP message is sent to a contact on a different domain, the client connects to its 'home' server, which then connects directly to the other contact's server, without intermediate hops.

Every XMPP entity needs and address, called JabberID (JID), containing a domain portion which can be resolved through DNS and a username portion that identifies the user itself. Additionally, when a client connects to a XMPP server, it can choose (or be assigned by the server) a resource identifier for that particular connection. As an example, the JabberID 'tiago@jabber.org/mobile' is divided in tree parts: username, domain and resource.

XML stanzas can be thought of as the basic unit of communication, similar to packets or messages in other network protocols, in XMPP. There are three kinds of stanzas (Message, Presence and IQ), that can carry different meanings, attributes and payload definitions.

These stanzas are exchanged asynchronously with other entities on the network. This event-driven approach has a number of advantages, such as real-time notifications and the ability to work around the need to continually poll for updated information.

XMPP reaches beyond the instant messaging realm; it is now applied to a range of applications, including gaming, social networking, VoIP, real-time collaboration and custom applications. XMPP technologies are hosted

in decentralised client-server systems with an unlimited number of servers thus providing security and scalability and there is no single point of failure.

2.8 Moving data across the Grid

gLite, the grid middleware currently installed in UMinho's Grid sites, has many different tools that can be used to move data across different devices.

A Storage Element (SE) provides uniform access to data storage resources in the EGI (most sites provide at least one SE), allowing a user or an application to store data for future retrieval. It may control simple disk servers, large disk arrays or tape-based storage systems.

SEs can support different data transfer and access protocols. In summary, the protocols supported in gLite 3.2 are [6]:

GSIFTP - a Grid Security Infrastructure (GSI) secure FTP protocol for whole-file transfers. It is responsible for secure file transfers to/from SEs. Every EGI site runs at least one GSIFTP server.

RFIO - offers direct remote access of files stored in the SEs in a secure and an insecure version

gsidcap - GSI enabled version of the dCache native access protocol.

file - used for local file access to network filesystems.

Most storage resources are managed by a SRM[13], a middleware service providing file handling capabilities. Any type of Storage Element in EGI offers an SRM interface except for the Classic SE, which was phased out. However, SRM implementations from different storage system may differ and offer different capabilities. There are different storage systems used in EGI. As an example we have: Disk Pool Manager (DPM), used for for SEs with disk-based storage only; CASTOR, designed to manage large-scale mass storage systems, with front-end disks and back-end tape storage; and StoRM. SRM hides the complexity of the resources setup behind it and allows the user to request files, keep them on a disk buffer for a specified lifetime, reserve in advance space for new files, and so on. SRM offers also

[13]Storage Resource Management Working Group `https://sdm.lbl.gov/srm-wg/`

a third party transfer protocol between different endpoints, not supported however by all SE implementations. It is important to notice that the SRM protocol is a storage management protocol and not a file access protocol.

Chapter 3

Using the Cluster to Scale the Grid

3.1 Introduction

In line with the R&D projects in which University of Minho is involved, a method to hasten the installation and administration process of an EGI site was developed. This method is based upon Rocks, a RedHat Enterprise Linux based distribution that aims at an easy installation and administration of computer clusters.

3.2 EGI roll

The EGI roll intends not only to make de installation of a Grid site easier, but also to simplify its administration. A former team mate, Bruno Oliveira, developed a set of features to provide administrators an intuitive and easy to use tool to perform the most common administration tasks, including management of the site's configuration, virtual organisations and software updates[26].

With hundreds of deployed clusters, Rocks' approach has shown to be quite easily adapted to different hardware and logical node configurations. However, the Rocks architecture and implementation contains a significant asymmetry: the graph definition of all appliance types except the initial FE

can be modified and extended by the end-user before installation. To address this administrative discontinuity between nodes and FEs, rolls were designed and implemented. Rolls provide both the architecture and mechanisms that enable the end-user to incrementally and programmatically modify the graph description for all appliance types. New functionality can be added and any Rocks-supplied software component can be overwritten or removed. This approach to cluster construction has allowed to shrink the core of the Rocks implementation while increasing flexibility for the end-user. Rolls are optional, automatically configured, cluster-aware software systems. Current add-ons include: scheduling systems (SGE, PBS), Grid Support, Database Support, Condor, just to name a few. Community-specific rolls can be and are developed by groups outside of the Rocks core development group.

3.2.1 Graph-based configuration

The Rocks toolkit (section 2.2) uses a graph-based framework to describe the configuration of all node types (known as appliances) that make up a complete cluster.

Rocks offers system administrators the ability to customise the packages to be installed by creating rolls, and to create dependencies between them using a graph-based framework. Some graph nodes are used to specify an appliance, which represents a particular computer node role or type. In order to be as flexible as possible, Rocks creates the kickstart on demand based upon a graph representation of the dependencies between packages or services[27].

The graph representations express both the set of packages to be installed and the individual package configuration. Each vertex of the graph represents a package or a service and its configuration. Both graphs and nodes are described by xml files. The graph file specifies the framework hierarchy where edges connect nodes to each other and each node file contains a list of Red Hat packages and optional configuration scripts to turn a meta-package into a final software deployment. Along with the packages configuration, the installation process can run pre- and/or post- install scripts that have access to a global configuration MySQL database managed by the FE that sup-

ports complex queries. The configuration of a cluster can then be thought of as a program used to configure a set of software, whose state, that represents a single instantiation of a cluster appliance, may be referenced by XML configuration code.

Taking the graph approach, nodes installation can be easily extended by creating a new graph that adds an arc linking to the extended node. Once the node file is created and packages are placed in the file installation hierarchy tree, other appliances derived from the newly created node can be added to the site.

As part of this work, the roll was changed to accommodate the recent changes in the middleware system. Previously, each gLite appliance used to be an extension of the Rocks compute appliance, as shown in figure 3.1. However, on the upgrade to gLite 3.2, it was required to adapt those appliances and create additional ones. Plus, some of the EGI appliances share common packages. For that reason, we decided to create a layer of gLite node types, that inherit from the traditional Rocks compute node, and group those node types to form the final EGI appliances. This new graph structure, which is easier to adapt and maintain, is depicted in figure 3.2.

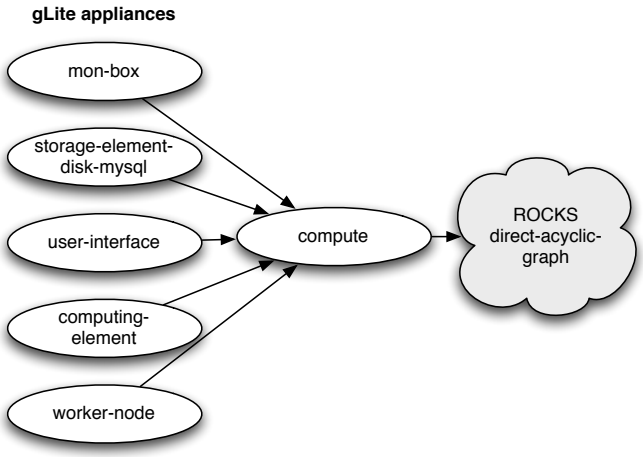


Figure 3.1: EGI roll old graph

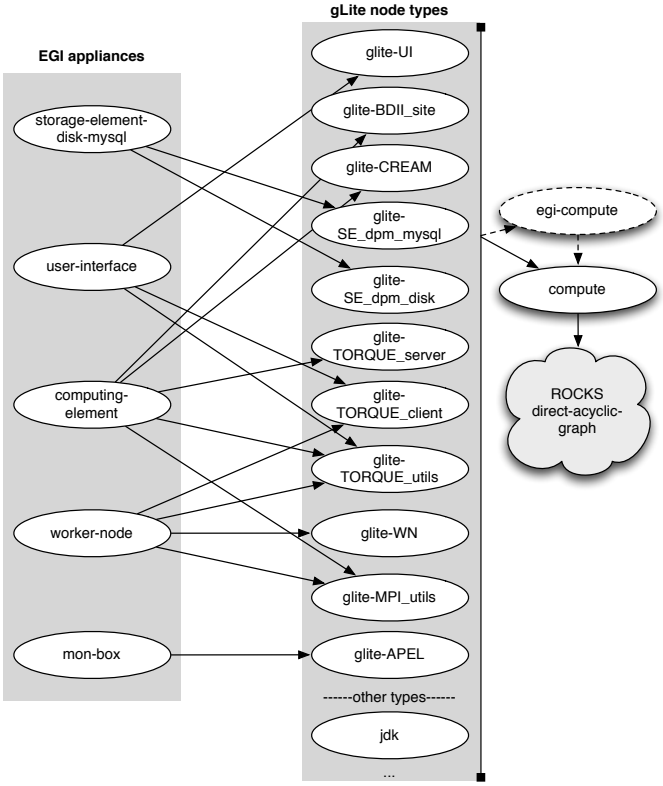


Figure 3.2: EGI roll new graph

3.2.2 Local software repository

One aspect that cannot be overlooked in the maintenance of a cluster is software updates. In a site with a significant number of nodes, this task can rapidly drain the available bandwidth and can represent a bottleneck in the system. The EGI roll gives system administrators the possibility to create a local repository of the software, so nodes can update themselves locally. This repository is created using the mrepo package[1]. The local repository holds a copy of the three major components in the EGI site: Scientific Linux, DAG and gLite middleware.

When installing the roll, system administrator can choose between the local repository or the normal repositories for the software. The creation of the local repository is time and hard-disk space consuming, but in the long run, this solution pays-off, since only one machine, the FE, has to fetch the

[1] mrepo homepage: `http://dag.wieers.com/home-made/mrepo/`

updated packages from the Internet, in a completely transparent automatic process, dealt by mrepo.

3.3 gLite RPM list

The *rocks* command, that is available in Rocks clusters, aims to merge under it all cluster administration tasks, that were scattered in several different commands. It is not about new commands, but a way to simplify the interface to tasks already offered, with an easy to use and understand syntax. The *rocks* command is based in verbs, like add, set, activate, list, among others.

Taking the *rocks* command as its base, the *egeecli* command was created to offer a clean and easy to use syntax to perform the administration tasks on a EGI site[26]. Particularly, it allows administrators to easily apply the frequently released updates of the gLite middleware, which provide new functionalities and bug fixes. That update process can take several steps, that involve not only the installation of the update via yum – or apt, in the case of a Debian environment – but can imply changes to the gLite configuration files, replication of these files and reconfiguration, via YAIM[2].

To aid administrators in the process, the *egeecli* command has adapted several verbs and created special objects that, together, automates the process. When a new update is released, administrators must add the update to the database, indicating the name of the meta-package, the version, a flag stipulating if this meta-package requires reconfiguration, release date and, optionally, a list of RPMs.

As part of this work, in order to aid administrators to obtain this list of RPMs, we created the *gLite parser*. This tool can be used to retrieve this list by parsing the gLite's release pages[3]. *gLite parser* is a script that takes the version of the gLite middleware as a parameter and creates a folder tree structure where each folder represents a gLite meta-package. Inside each

[2]YAIM is a configuration tool that executes the essential steps in order to prepare and configure the machine according to its role or appliance type. YAIM project website: `http://yaim.info/`

[3]gLite releases for version 3.2: `http://glite.cern.ch/R3.2/`

meta-package folder, other folders exist - one for each release - containing the RPM list in text form.

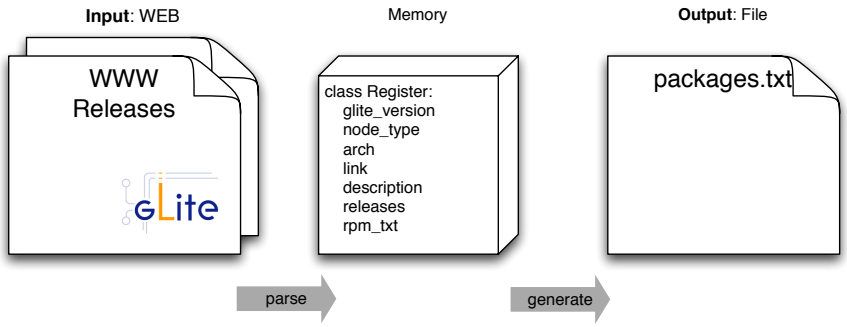


Figure 3.3: gLite parser workflow

As depicted in figure 3.3, the tool starts by parsing the releases web page for available meta-packages and then, for each one, parses its release page for released versions. For each version, the tool detects the list of the RPM packages that belong to this meta-package, and also any special information, such as if reconfiguration of the service provided by the meta-package is needed, or the priority level of this particular release.

The administrator can then use the desired meta-package/release to pass as argument when adding a new update, or to, for instance, retrieve all, or some, RPMs. This last option, is useful to recreate the EGI roll with the latest releases.

The parser, written in Python, is based on the HTML/XML parser BeautifulSoup[4], designed for quick turnaround projects like screen-scraping. BeautifulSoup simplifies the parsing task by turning HTML into a navigable parse tree.

While parsing the HTML pages, the tool fills a data structure, composed by instances of the class Register (see table 3.1).

Although, this tool has some limitations because it is based on the HTML structure of the gLite releases page, which may change and break the existing code. At the time of writing, the site structure doesn't fulfil the XHTML standards, making the parsing harder and leading to a more complicated code.

[4]BeautifulSoup homepage: `http://www.crummy.com/software/BeautifulSoup/`

Variable	Type	Description
glite_version	string	version of gLite. ex: 3.2
node_type	string	gLite node type. ex: glite-UI
arch	string	version architecture. ex: x86_64
link	string	url of the release page
description	string	release description. ex: gLite Worker Node (WN)
releases	dictversion:rpms	Map release-url of the RPM site
rpm_txt	dictversion:txt	Map release-url of the RPM list in .txt

Table 3.1: The class Register holds information about each gLite node.

3.4 Integration

The main goal of this study is to obtain an integrated management resource that handles different computing mechanisms in an easy and seamless way.

In order to join these technologies together, these dots have to be connected, filling the existing gap and creating a technology bridge as depicted in figure 3.4.

Every EGI site has to obey to certain rules, respecting the minimum requirements set by the project, and provide a set of control mechanisms. These components, also known as elements in gLite, are installed in different machines. As described in section 3.2, this is performed in an easier way using the Rocks Clusters mechanisms.

Traditionally, the structure of an EGI site is almost static, composed by a set of machines. This integration method allows the administrator to easily expand the computing resources, according to the needs, using virtualisation technologies. New clusters can be easily deployed without needing to modify the physical structure of the site, severely increasing its scalability.

One of the requirements of the CROSS-Fire project is to develop the

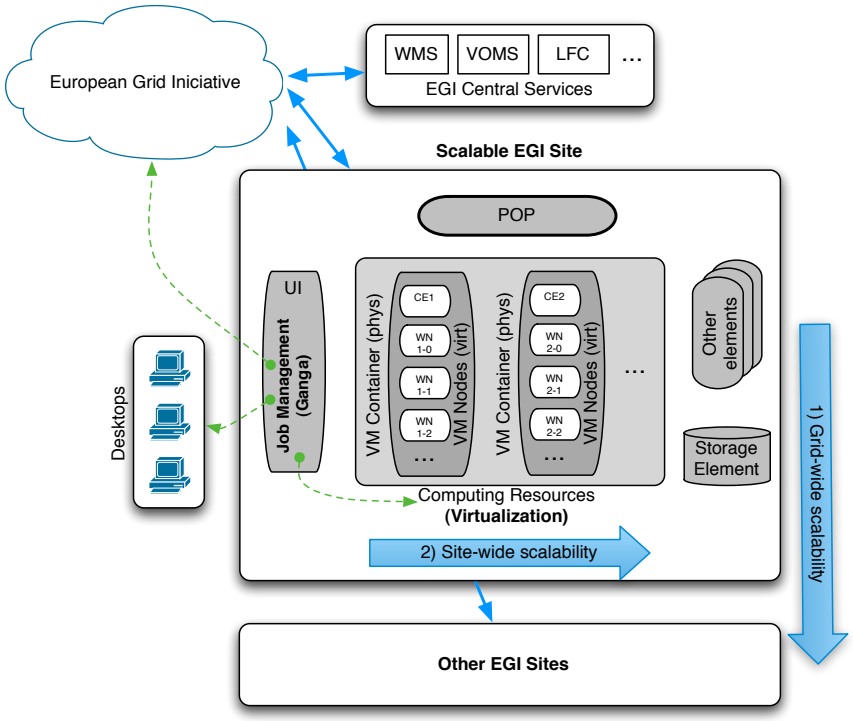


Figure 3.4: EGI roll scalability.

scalability and flexibility of the system, to permit new institutions to rapidly integrate the project.

Summing up, the general ecosystem can easily scale in two different dimensions. The first one, referred as Grid-wide, allows multiple sites to be efficiently deployed by using the mechanisms provided by Rocks Clusters and, more specifically, solutions like the EGI roll. The second vector, referred as Site-wide, allows the site itself to grow within its boundaries, it terms of the services it provides. This idea is expressed in figure 3.4.

The Job Management software, described in section 2.4, is an important component of the platform that allows the user to easily run his jobs in different environments using the same job description. More than just running the job, it permits monitoring the task, retrieving the outputs and selecting the destination.

Chapter 4

Connecting users across Grid sites

Advanced computing infrastructures are complex environments where the user has limited interference. The user can choose between traditional infrastructures or several other options that don't impose significant prior investments, such as the Grid, virtualised Clouds, and even by personal systems.

The User Domains platform, presented in [1] tries to leverage resources from multiple providers in an efficient user-level infrastructure overlay. The authors proposed the Domain abstraction to manage user-level computing capacity available in multiple sites. The resources are unified in a personal overlay infrastructure to allow the creation of consistent and flexible computer environments largely independent of any particular resource provider. This overlay is unique for each user and presents an interface system that mediates the connection between all the components of the system. In this context a Domain is a territory governed by a single ruler, it represents the computing elements the user has access to locally or in various resource providers.

However, NAT devices are in widespread use, so addressing the problem of accessing resources inside them is fundamental in a global access platform. We are currently evaluating SPM in more complex scenarios, measuring the interference of SPM in the Ethernet network overlay and in the disk block

cache.

The ability to present users with a set of the available resources and to provide an interface to access and explore those resources is fundamental in any computing system. In [1] the cluster job manager provided a list of available resources, the cluster login nodes served as the interface to the resources, and users connected to the system remotely using ssh sessions and port remapping.

4.1 Simple Peer Messaging (SPM)

SPM is a light-weight peer-to-peer communications library, developed in the context of User Domains, that is capable of: i) discovering related live entities using a global presence and messaging system and ii) building efficient and reliable communication links between any entities that are successfully registered in that system, including those located behind NAT devices.

To allow remote infrastructures to be connected by User Domains in a single user-mode overlay, they must be aware of the existence of each other and must be able to effectively communicate so that the performance is not exceedingly impaired by the network links.

The SPM library is a peer-to-peer communication system that was devised to bridge the gap between the dispersed network segments providing an overlay that runs entirely in user-mode and where each infrastructure is autonomous and no central control exists. SPM delivers a network layer with two distinct functions: i) the gathering of information about live peers and basic session establishment and ii) the direct and reliable interconnection of the peers using NAT traversal techniques.

Figure 4.1 presents SPM as used in the context of User Domains. In this example an interface Domain is connected to local single-node Domain Group and a cluster-based multi-node Domain Group. It connects intra group System Agents (SAs), storage functions, Ethernet overlay (VM Network) and inter group SAs (SPM interconnection).

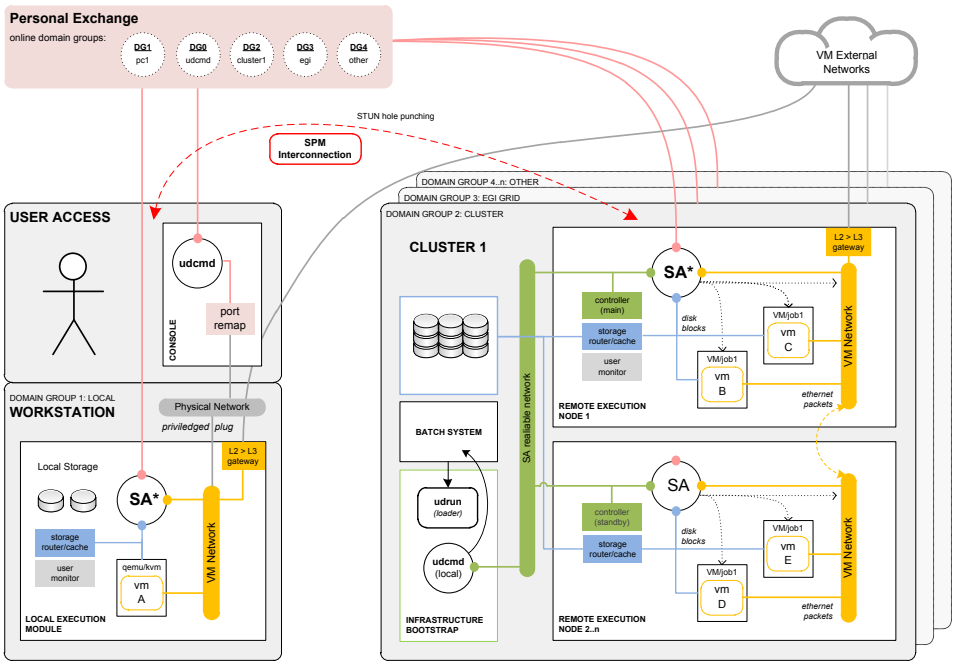


Figure 4.1: User Domains spanning multiple heterogeneous infrastructures with SPM

4.1.1 Personal Exchange for Domain Groups

Personal Exchange is a basic presence and messaging system between remote agents. It provides a central point of aggregation for each user, with a technology agnostic panorama of the computing resources gathered from multiple locations and infrastructures. The Personal Exchange hides the details of each type of system by including infrastructure proxies, which cooperate with each other to provide a common gateway to resources in the infrastructures the user has access to. The Personal Exchange allows Domain Group awareness, basic communication and sending traffic between SAs in different Domains Groups, including control requests, Ethernet network and disk blocks.

SPM implements the Personal Exchange using the widely deployed Extensible Messaging and Presence Protocol (XMPP), an open standard technology for real-time interaction (see section 2.7). SPM can take advantage of any XMPP compliant service such as the massive communication services

associated with Google Talk, readily available to Gmail users. Nevertheless, personal servers can be used to create private systems.

4.1.2 Point-to-Point links between remote domains

As described in section 2.6, traversing NAT can be challenging. SPM resorts to Interactive Connectivity Establishment (ICE) to implement NAT traversal. It defines an offers/ answer model that takes advantage of several different NAT traversal strategies until a working one is found. It offers multiple IP addresses and ports as connection candidates, which are then tested with peer-to-peer connectivity checks. The IP addresses and ports included in the candidates and the connectivity checks use the revised STUN for NAT specification and TURN. The purpose of ICE is to discover which pairs of addresses will work, by systematically trying all possible pairs until it finds one or more that work.

Initially the end-points have no information about the network topology and they may be behind multiple tiers of NAT, or may be directly connected. ICE assumes that two endpoints wanting to communicate are able to communicate indirectly via some signalling protocol that allows the exchange of messages. ICE is used to discover enough information about the topology to find one or more paths by which they can potentially communicate. Each agent has a set of candidate transport addresses that can be used to communicate with the remote's candidate transport addresses. In practice, however, most combinations will not work. For instance, L and R are behind different NATs, so their directly attached interface addresses are unlikely to be able to communicate directly. The low level interconnection always uses UDP, but there is the option to establish reliable connection using the PseudoTCP emulation of TCP over UDP. This protocol is not a complete TCP substitute, as there are no timeouts on SYNCTL_CONNECT message and there are no implicit shutdown semantics. But those limitations can, in fact, be advantages in loosely connected environments.

4.1.3 Application Programming Interface

SPM can be used independently from User Domains through a very simple Application Programming Interface (API). Although peer-to-peer oriented, underlying SPM is a client/server model of communication, in which any node can initiate a connection with any other node. An application interested in receiving requests on a topic must join the group that represents that specific topic and wait for other clients to connect to it. Once a client initiates a conversation with the servers that are registered in a topic, a separate connection is established using NAT traversal as required.

Listing 4.1: SPM main API functions

```
1  /* connect to and disconnect from spm */
2  spm_t * spm_init( char* jid, char* pass );
3  void spm_done( spm_t* h );

5  /* join, leave and enumerate group members */
6  int spm_join( spm_t* h, char* group, char* name );
7  char** int spm_enum( spm_t* h, char* group, int * n );
8  int spm_leave( spm_t* h, char* group, char* name );

10 /* establish ICE sessions */
11 spm_link_t* spm_wait( spm_t* h, long timeout_ns,
12     int * reliable );
13 spm_link_t* spm_initiate( spm_t* h, char* group, char* name,
14     int * reliable );
15 void spm_closelink( pm_link_t* l );

17 /* send data through ICE sessions */
18 int spm_send( spm_link_t* l, void* buff, int sz );
19 int spm_recv( spm_link_t* l, void* buff, int sz,
20     long timeout_ns );
```

Listings 4.2 presents the prototype functions of the basic API required to:

1. connect and authenticate in the system
2. join and leave interest groups
3. establish network connections with peers in those groups
4. directly send and receive data

The API also includes functions that allow handling communication using event handlers, which are more adequate to deal with multiple simultaneous connections.

4.1.4 Intra and Inter Domain-Group Routing

User Domains' network routing scheme is independent of SPM, which is used in this context only to interconnect different Domain Groups. Every SA is a network router and each Domain can contact any other by routing the network messages to the intended destination using the available interconnections. SPM provides point-to-point links between tasks, which are maintained as a set of active connections established on request between the Domain Groups. These connections are created and destroyed by SAs according to the destination of the messages. Like other infrastructure drivers, SPM is required to provide a list of interconnection peers and a database (Exchange) to list SAs capable of interconnecting different network segments. Whenever the destination is unknown or is unreachable the SAs trigger a destination discovery process, gathering all addresses accessible and the paths to be followed in each case, using a direct routing strategy afterwards.

4.1.5 SPM Port Mapper utility

While SPM was developed in the context of User Domains to interconnect multiple remote locations, we believe there are several usage scenarios in which it can be very useful. One example of such scenarios is an SPM application we implemented to serve as a generic port redirector, similar to the *netcat* utility. It performs network tunnelling between NAT environments, with which users can publish services and establish tunnels between multiple network services without needing to use public IP addresses.

4.2 Connecting services among Grid sites

One of the objectives of this work was to develop a solution to connect different services that were not directly connected to the Internet (don't have

a public and accessible IP address). These private addressed machines, residing behind NAT's, need special techniques in order to establish a connection between them. There are many known techniques, described in section 2.6, that can be used to accomplish this goal.

The first strategy, NAT reconfiguration, involves explicit signalling between applications and NAT devices, briefly described in section 2.6.1, is out of the scope of this work, since it is not a real solution for our problem. Although these techniques are frequently used by popular p2p applications in small and home networks and supported by most of the common Integrated Service Routers (ISRs), the network environments where this work is inserted is normally controlled by strict security policies. For this reason, this NAT traversal strategy was excluded form the solution, although it could be useful under certain conditions.

Relaying, described in section 2.6.2, consists in traversing NAT devices using a relay server. As previously mentioned, this technique can be useful as a fallback, when everything else fails, or on very specific scenarios. The fact that all the traffic passes through a relay server is a big performance penalty that excludes this strategy from our solution.

Techniques such as hole punching rely on a third party server in order to establish the connection. This is can be a feasible way to overcome the problem, however, different devices and topologies require employing different mechanisms. It is necessary to get information about each topology to apply the correct technique. This is the may reason why we didn't choose to test different hole punching techniques.

Section 2.6.3 introduced a novel technique for NAT traversal that doesn't rely on a third party server. This fact brings many advantages when compared with other strategies.

Among several techniques that were studied and analysed, we decided that this simple and precise method could be a solution for our problem. This 'autonomous nat traversal' strategy[18], were implemented by the authors on the *pwnat* tool[1], a stand-alone implementation of autonomous NAT traversal. Pwnat is a simple program, openly released for nix operating sys-

[1]pwnat project homepage, by Samy Kamkar: `http://samy.pl/pwnat/`

tems, that must be run on both endpoints of the communication, one acting as a client and the other as a server[2]. It allows one ore more clients behind NATs to connect to a server behind a separate NAT. As written by the authors, 'There is no middle man, no proxy, no 3rd party, no UPnP/STUN/ICE required, no spoofing, and no DNS tricks'.

However, this method is not infallible. To evaluate that, the authors developed and released the NAT-Tester framework[3], which can be used by any user and allowed the authors to gather information about common NAT devices around the Internet. The main results of these studies were published in 2010[18], where it was concluded that the autonomous NAT traversal technique 'works extremely well if only one peer is behind NAT and virtually never if both peers are behind NAT'.

We have tried this alternative in two different scenarios: from a publicly addressed, to a host behind NAT; and from two hosts behind NAT. On the first scenario, the evaluation was successful - We were able to connect to the NATed device. The second one is more complicated - even running pwnat on both machines, they were not able to make a successful connection (this is the harder scenario, as concluded by the evaluations made by the authors of the application).

The usage of pwnat is fairly simple, as you can see bellow, however is didn't show to be as robust as our project requires because it fails in one of the most important scenarios - when you have two agents behind different NATs.

Listing 4.2: SPM main API functions

```
1 % download and compile pwnat on both hosts

3 wget http://samy.pl/pwnat/pwnat-0.3-beta.tgz
4 tar xvzf pwnat-0.3-beta.tgz
5 cd pwnat-0.3-beta
6 make
```

[2]For clarification, on this section, the client and server terms refer to the hosts running the pwnat application as client or server, respectively.

[3]NAT-Tester homepage, by Andreas Müller and Andreas Klenk: `http://nattest.net.in.tum.de/`

```
% run pwnat in server mode on host 1
(host 1)# ./pwnat -s

% run pwnat as client on host 2
(client)# ./pwnat -c 8000 <pwnat.server.com> google.com 80
```

Once again, we can understand that there isn't a fully successful solution for the NAT traversal problem, but each new alternative, like this one, potentially increases the final success rate.

4.2.1 Using ICE for NAT traversal

In order to be able to connect different devices in separate private networks, behind NAT devices, we chose to use the ICE method. RFC 5245 (produced by IETF in April 2010) describes ICE as a technique, rather than a protocol, for NAT traversal established by the offer/answer protocol. ICE does not use a single technique for traversing NAT, but takes advantage of a few existent other methods in an integrated and simpler way. It incorporates hole punching, relaying and even NAT reconfiguration techniques (see section 2.6). This mechanism was described in deeper detail in section (2.6.4).

In a typical ICE deployment, we have two endpoints, known as agents, that want to communicate. They are able to communicate indirectly via some communication protocol (such as SIP or XMPP), in order to exchange some information about themselves. At the beginning of the ICE process, the agents have no information about their topologies. They might be behind multiple tiers of NAT, or might not even be behind one. ICE facilitates this topology discovery process to potentially find one or more paths by which they can communicate.

Figure 4.2 is an example of a typical ICE deployment. The two endpoints (labelled L and R) are behind their own respective NATs though they may not be aware of it. The type of NAT and its properties are also unknown. Agents L and R are capable of exchanging session establishment information through a broker (typically a XMPP or SIP server) in order to set up a p2p session between L and R.

In addition to the agents, a signalling server and NATs, ICE typically

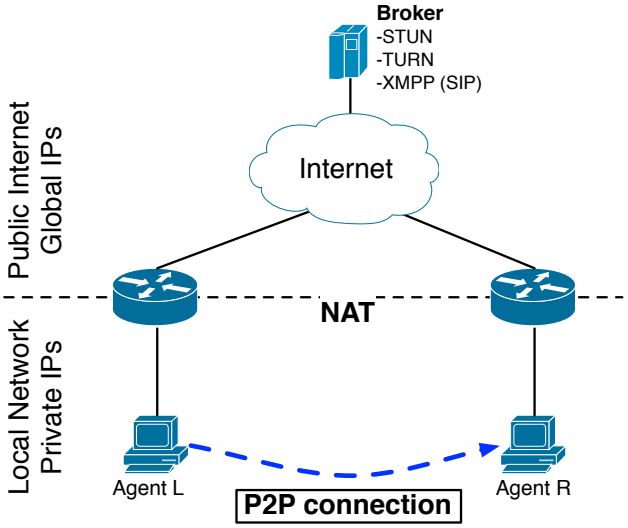


Figure 4.2: Typical ICE deployment scenario

relies on STUN or TURN servers as well (each agent can have its own STUN or TURN server, or they can be the same).

Each agent has a variety of candidate transport addresses (combination of IP address and port for a particular transport protocol) it could use to communicate with the other agent, including (see figure 4.3):

- A transport address on a directly attached network interface (a ẗhost candidateäddress)
- A translated transport address on the public side of a NAT (a s̈erver reflexiveäddress)
- A transport address allocated from a TURN server (a r̈elayed address)̈

Each agent potentially has a set of candidate transport addresses that can be used to communicate with the remote's candidate transport addresses. In practice, however, many combinations will not work. In this example, for instance, L and R are behind different NATs, so their directly attached interface addresses are unlikely to be able to communicate directly (this is why ICE is needed). The purpose of ICE is to discover which pairs of addresses will work, by systematically trying all possible pairs (in a carefully sorted order) until it finds one or more that work.

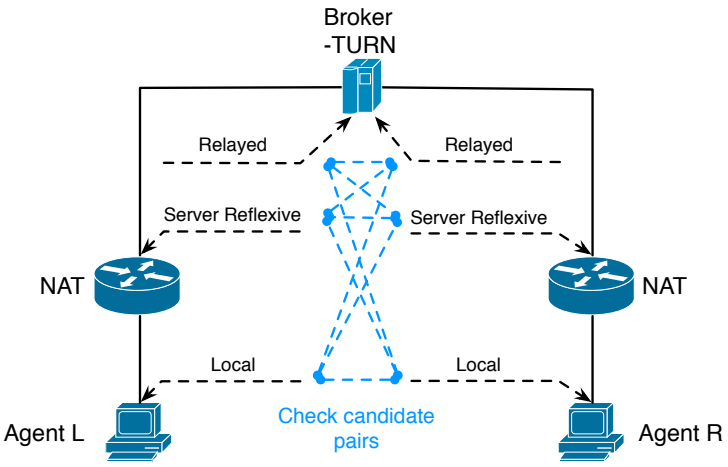


Figure 4.3: Main transport address candidates on ICE

In order to execute ICE, an agent has to identify all of its address candidates. Naturally, one viable candidate is a transport address obtained directly from a local interface (host candidate). Next, the agent uses STUN or TURN to obtain additional candidates. These come in two flavours: translated addresses on the public side of a NAT (server reflexive candidate) and addresses on TURN servers (relayed address). When TURN servers are utilised, both types of candidates can be obtained from the TURN server. If only STUN servers are utilised, only server reflexive candidates are obtained from them. In other words, a TURN server generally provides a STUN and TURN service.

Once each agent has gathered all of its candidates, it sorts them in descendant priority order and sends them to the remote agent over the signalling channel (i.e. using the XMPP service). At the end of this process, each agent has a complete list of both its candidates and its peer's candidates. Next, it pairs them up, to check which pairs work (simplified on figure 4.3). High-priority candidate pairs are checked first, followed by lower-priority ones. Summarising, the basic principle is simple:

1. Discover the candidate transport addresses
2. Sort the candidate pairs in priority order
3. Send checks on each candidate pair in priority order

4. Acknowledge checks received from the other agent

At the end of this handshake, both L and R know that they can send (and receive) messages end-to-end in both directions.

There are a few open implementations of this protocol. After evaluating all of their potentialities and limitations, we chose to use *libnice*[4], a free implementation of the ICE specification written in C, based on the GLib library.

Using the (sometimes short) available description of the library and the valuable help of its community, we started by developing a simple application to test the connection between two separate NATed hosts.

When using libnice, NiceAgent is the main object that takes care of everything relating to ICE. It takes care of discovering local candidates and doing connectivity checks to create a stream of data between both peers. Listings 4.3 details the main structure of libnice usage[5]:

Listing 4.3: Simple example on how to use libnice

```
1  // Create a nice agent
2  NiceAgent *agent =nice_agent_new(NULL,NICE_COMPATIBILITY_RFC5245
       );
3  // Connect the signals
4  g_signal_connect(G_OBJECT(agent),"candidate-gathering-done",
5                   G_CALLBACK(cb_candidate_gathering_done),NULL);
6  g_signal_connect(G_OBJECT(agent),"component-state-changed",
7                   G_CALLBACK(cb_component_state_changed),NULL);
8  g_signal_connect(G_OBJECT(agent),"new-selected-pair",
9                   G_CALLBACK(cb_new_selected_pair),NULL);
10 // Create a new stream and start gathering candidates
11 stream_id = nice_agent_add_stream (agent, 1);
12 nice_agent_gather_candidates (agent, stream_id);
13 // Attach to the component to receive the data
14 nice_agent_attach_recv(agent,stream_id,1,NULL,cb_nice_recv,NULL)
       ;
15 // Wait until the signal candidate-gathering-done is fired ...
16 lcands = nice_agent_get_local_candidates(agent, stream_id, 1);
17 // Send local candidates and set the peer's remote candidates
```

[4]libnice project homepage: `http://nice.freedesktop.org/`

[5]Similar example can be found on the libnice API: `http://nice.freedesktop.org/libnice/NiceAgent.html`

```
18 nice_agent_set_remote_candidates(agent,stream_id,1,rcands);
19 // Wait until the signal new-selected-pair is fired ...
20 // Send our message!
21 nice_agent_send(agent,stream_id,1,sizeof(buffer),buffer);
```

In order to succeed, each ICE agent needs to find its own candidates and send them to the remote agent, as commented on line number 17 of the code above.

Initially, we accomplished that by writing the *NiceCandidate* structs to file and exchanging them between the agents using a FTP/SSH server. However, after these successful initial tests, we found the need to create a more sophisticated method for this control information exchange. After doing some research and questioning the community, and excluding some other unsuitable possibilities, we got before two different solutions for the problem. Section 4.2.2 describes the solution for this problem.

NiceCandidate is the structure that carries relevant information about transport candidate addresses[6]. The main fields are described in listings 4.4:

Listing 4.4: NiceCandidate structure

```
1 struct NiceCandidate {
2   NiceCandidateType type;     //type of candidate
3   NiceCandidateTransport transport;     //transport being used
4   NiceAddress addr;     //address of the candidate
5   NiceAddress base_addr;     //base address used by the candidate
6   guint32 priority;     //priority of the candidate
7   //...
8   TurnServer *turn;     //TURN settings for relayed candidates
9 };
```

4.2.2 Using XMPP for control information exchange

XMPP, previously described in section 2.7, is an open XML protocol for real time messaging, presence and request response services. As described in section 4.2.1, we needed a simple and versatile way of exchanging the ICE information between agents.

[6]more detailed information on the API: `http://nice.freedesktop.org/libnice/libnice-NiceCandidate.html`

XMPP has shown up to be the right way of doing it. In addition to its XML messaging main feature, XMPP has the concept of presence and contact lists. As in any common chat application, each user can have a list of contacts, check his presence status and send/receive messages. Moreover, it even deals with the authentication concerns. Any user can freely take advantage of the infrastructure and adapt it to his needs.

After choosing the right protocol for this simple information exchange, we decided to start building a basic application to accomplish our needs. One of the advantages of choosing a popular open standard is that you can get lots of support and examples. That's the case with XMPP[7]. There are plenty of projects going on and, in the homepage, you can find a list of servers, client software and libraries.

We decided to try Strophe[8], a collection of libraries for speaking the XMPP protocol. Strophe comes in two flavours: Strophe.js, a JavaScript implementation targeting browser-based clients and libstrophe, a minimal C library for XMPP clients and components, designed for both POSIX and Windows systems. As the rest of our application was developed in C code, we coherently decided to go with libstrophe.

XMPP is a very simple protocol. Yet, there are many components, modules and extensions that can be used, depending on your product needs.

4.2.3 Simple Peer Messaging (SPM)

SPM is the application that gathers these functionalities together, in order to tackle our main communication goal: the ability to communicate among different grid sites. It uses the libnice ICE implementation to establish a connection across NAT devices and libstrophe XMPP library to exchange control information.

So, SPM's main functionality is to establish a transport connection between two hosts. After managing it, the application has to be fed with data, the raw material that needs to be transmitted. We decided to use POSIX sockets to deal with the data input and output of SPM. The high-level architecture of SPM is described in figure 4.4.

[7]XMPP homepage: `http://xmpp.org`

[8]Strophe homepage: `http://strophe.im`

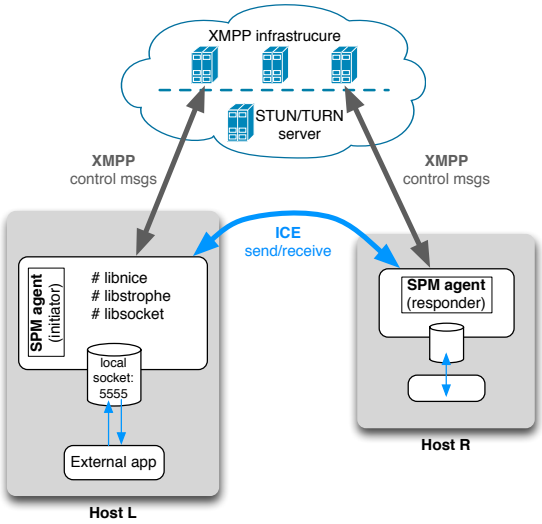


Figure 4.4: SPM high-level architecture. Two hosts communicating through ICE

Summing up, SPM starts by registering itself (the user) in the XMPP service. This creates a pool of contacts that can be seen as a set of resources that are available at some point. Then, when one of those contacts is chosen, se communication and negotiation starts. An invitation to start a ICE conversation is sent to the remote contact. This negotiation involves exchanging ICE agent credentials and candidate addresses. When this ICE connection is established, SPM opens a local socket for getting (and returning) data from (to) an external application. This is achieved by opening a local socket. That data that comes from the local socket is sliced and sent through the ICE connection, using the appropriate libnice call. The same applies in the reverse direction. As data arrives from the remote host through ICE, it is continuously passed to the local socket. Briefly, this is how data is passed from one host to the other.

Besides its simple high level architecture, SPM has a quite complex sequencing. It is responsible for managing and sequencing three different sources/destinations of data: the ICE connection, the XMPP communications and a local socket. Figure 4.5 represents that sequence diagram. As you can read from it, SPM starts by asking ICE for the host's local creden-

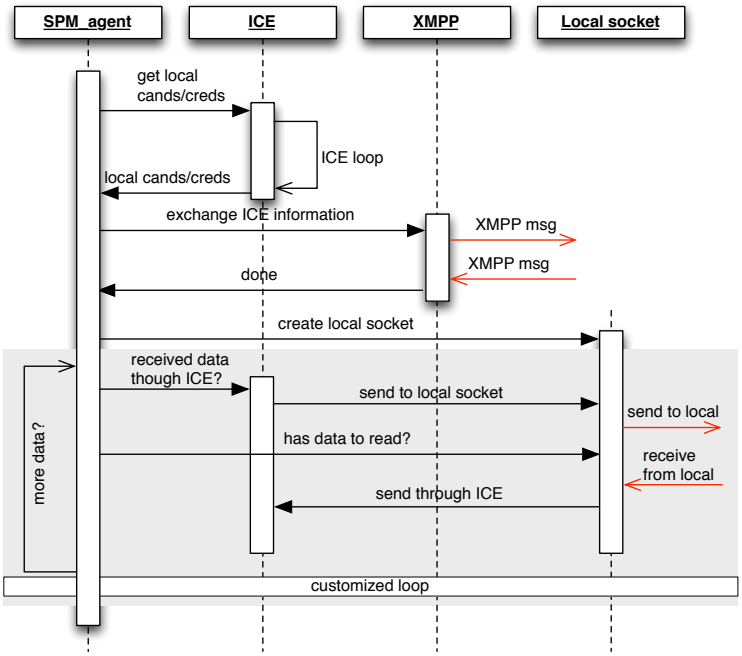


Figure 4.5: SPM sequence diagram. Involves ICE, XMPP and a local socket

tials and transport candidates. This operation involves asking the TURN server for the relayed address candidate. After ICE gets that information, the loop is stopped and the credentials and local candidates are returned.

The XMPP exchange is then started. The local agent (initiator) invites the remote agent (responder) to start a ICE conversation. After it is accepted, agents exchange each local credentials and transport candidates. When this process is achieved, the XMPP loop stops and the local socket is created. This socket is used to get data from any external application that is therefore sent though ICE (the reverse also applies).

After creating the local socket and accepting a connection from the external application, a customised loop is run. In each iteration of this loop, a set of checks is made. Firstly, it starts by checking if any data has been received through ICE. In that case, that data is immediately written to the local socket. Then, using select, SPM checks if there is data to be read from the local socket. In case there is, that data is sent through ICE.

Figure 4.6 describes in greater detail the XMPP communication, that

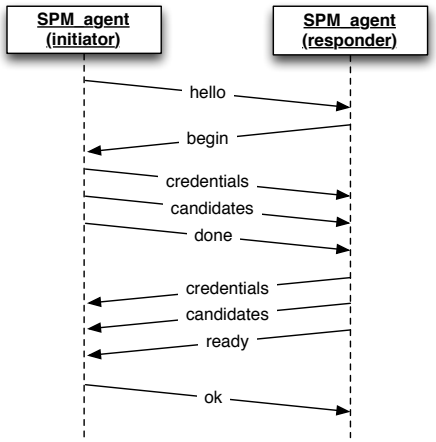


Figure 4.6: XMPP handshake for ICE conversation establishment

works as an handshake to establish a ICE connection. We decided to create a very simple protocol between the two agents where credentials and transport candidates are exchanged using XMPP messages.

Listings 4.5 shows three examples of messages exchanged through XMPP (credentials, candidates and done).

Listing 4.5: Example of three messages sent the XMPP handshake

```
1  % credentials message
2  <message id="myICE" to="tiago2@jabber.org" type="normal">
3      <subject>credentials</subject>
4      <body>e3YZ /Vu0fnsreWZrkyjii/JYHt</body></message>

6  % candidates message
7  <message id="myICE" to="tiago2@jabber.org" type="normal">
8      <subject>candidates</subject>
9      <body>AAAAAAAAAAACAIevrBBUgwAAAAAAAAAAAAAAAAAAAg...</body>
10     <body>AQAAAAAAAAACANyjwYgTNwAAAAAAAAA////AAAAAg...</body>
11     <body>AwAAAAAAAAACAMphwYgTvwAAAAAAAAA////AAAAAg...</body></
           message>

13 % done message
14 <message id="myICE" to="tiago2@jabber.org" type="normal">
15     <subject>done</subject></message>
```

Chapter 5

Application of main results

5.1 University of Minho Grid sites

The core research computing infrastructure at UMinho deployment environment is based on the Rocks toolkit.

In order to offer its researchers a Grid network suited to support different types of scientific applications and to contribute with developments for applying it to different scientific fields and communities, UMinho became a member of the EGEE South West federation production infrastructure back in 2007. UMinho's EGEE first site (named DI-UMinho) was installed and configured with the gLite middleware based on Scientific Linux 3.

Later, a second site was installed to support the grid activities taking place at UMinho (named UMinho-CP). This site responded to the specificities of the Civil Protection (CP) projects in which the UMinho was involved.

UMinho-CP has evolved since its first deployment, following the needs of the projects where it participates and the evolution of the infrastructure itself. Figure 5.1 depicts the actual state of the site, where SVMs 1 to 5 are virtualisation servers running VMware ESX Server.

Traditionally, each cluster based on Rocks is built around a private VLAN, where the FE acts as a gateway. However, in this scenario, some cluster nodes may need to be directly connected to the Internet in order to provide different Grid services. The private network (private VLAN in the figure) is used to deploy and maintain the cluster, mastered by the FE.

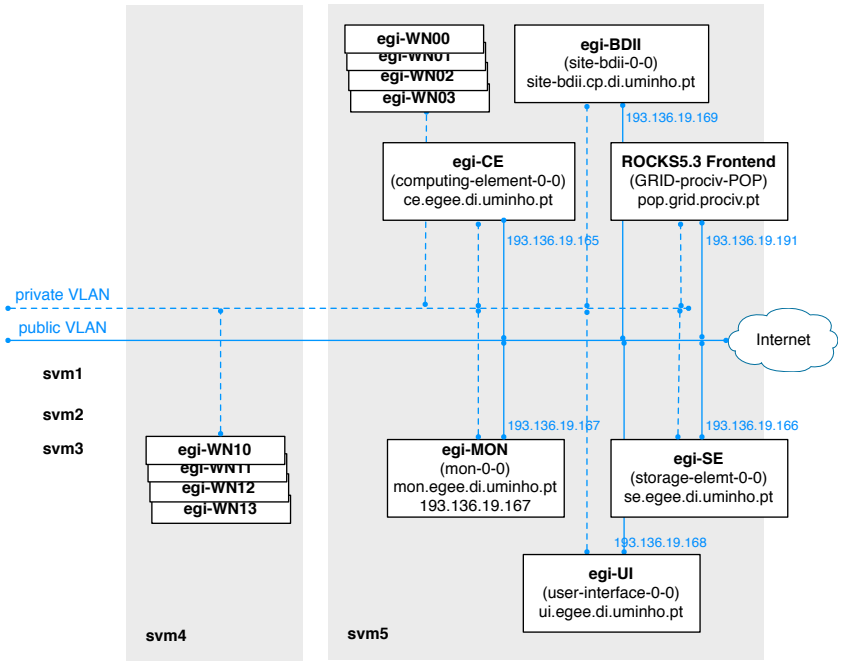


Figure 5.1: UMinho-CP site infrastructure

Additionally, every node except the Worker Nodes (WNs) is also connected to the Internet, addressed with a public IP. That Internet connection is indicated by the public VLAN and public IPs in the figure.

The architecture model of each site comprises a set of WNs in a tightly coupled LAN connected by a private Ethernet switch and several different server components connected both to the internal network and to the Internet.

A Virtual Organization (VO) refers to a dynamic set of individuals or institutions defined around a set of resource-sharing rules and conditions. All VOs share some commonality among them, including common concerns and requirements, but may vary in size, scope, duration, sociology, and structure.

5.2 SPM - performance evaluation

NAT traversal evaluation has focused primarily on the ability to perform peer interconnection [12, 16] and there is little information about the per-

formance achievable by NAT traversal techniques. However, User Domains platform's performance is highly dependent on the network characteristics, so further evaluation was required to study the network bandwidth of NAT traversal in multiple scenarios of interest[1].

This performance evaluation compares the bandwidth achieved while transmitting 1GB of data using SPM over PseudoTCP/UDP to the bandwidth that is achieved by the *iperf* benchmark directly over UDP[1].

Since the *iperf* benchmark follows the client/server model and cannot connect to a server behind a NAT devices autonomously, in the tests that required NAT traversal the routers were configured to forward the required ports to the testing hosts, which should introduce a small processing overhead on the remap. But a UDP only program is also relieved from providing reliable PseudoTCP service, which itself also introduces an overhead in the libraries supporting SPM. So, while this comparison is not strictly accurate, we are convinced that the obtained results may be used to evaluate the applicability of the approach.

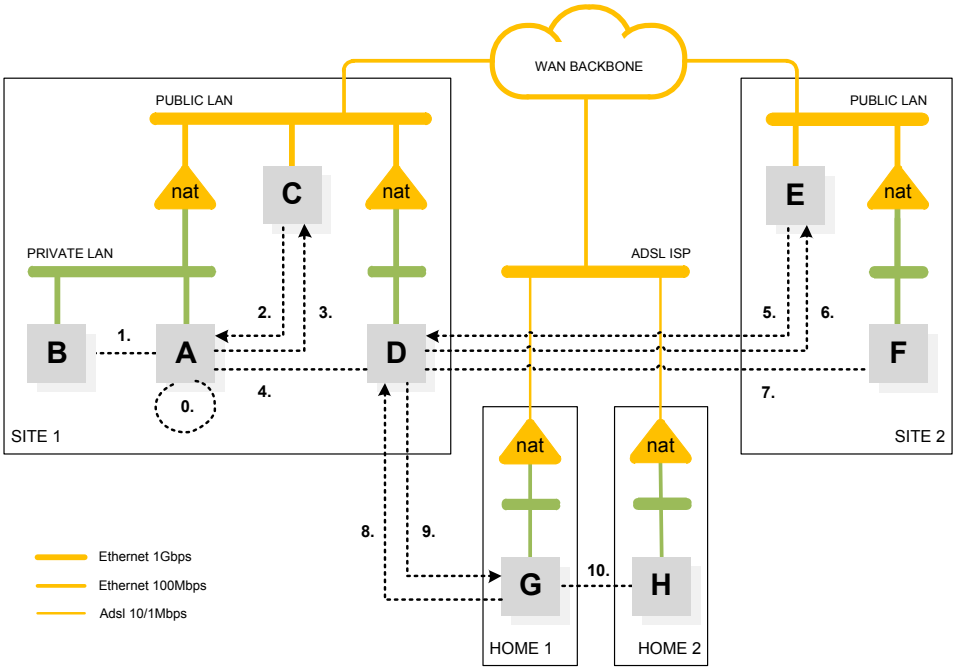


Figure 5.2: Test setups to be used as reference

[1]iperf project homepage: `http://sourceforge.net/projects/iperf/`

The tests were performed between 4 different sites: the high bandwidth site1 and site2 sites are typical HPC facilities and the low, and asymmetric, bandwidth sites home1 and home2 are domestic (see figure 5.2). The test setups included three categories: LAN, high speed WAN and domestic WAN, then further divided in eleven different network topology scenarios, including configurations:

1. without NAT (0, 1)
2. with NAT on one side (2, 3, 5, 6)
3. with NAT on both sides (4, 7, 8, 9, 10)

5.2.1 Test-bench configuration

The testing workbench was based on nodes containing two six-core Intel Xeon X5650 processors and 24GB of RAM interconnected by Gigabit Ethernet, installed with the Linux OS kernel 2.6.18-194 (CentOS 5.4). Two types of routers were used: router servers based on compute nodes similar to the ones above in the two main sites and domestic router appliances in the domestic environments. SPM hole punching and *iperf* tests were performed using an UDP block size of 4K bytes, while the network MTU was kept in the default value of 1500. The results presented are the median of 5 runs of each performance test and all processes were pinned to a specific processor core.

5.2.2 Experimental results

Based on the evaluated configurations, we can conclude that SPM was able to establish direct connection to remote nodes using NAT hole punching in every scenario.

Table 5.1 lists the bandwidth achieved in the network configurations presented in figure 5.2. The Efficiency column shows the ratio between *SPM* and *iperf*, representing the overhead of both our library and its dependencies and the NAT traversal process. From the results, one can conclude that when connecting different compute nodes the bandwidth available in SPM

is always within 87% of the raw network performance. Plus, this level of performance is stable from the low bandwidth domestic ADSL connections up to the gigabit LAN configurations. Given the intrinsic overhead of the SPM library, one can also conclude that the NAT hole punching process is not exceedingly penalising.

Setup	**SPM (Mbps)**	**iperf (Mbps)**	**Efficiency (%)**
0.	6.012,10	7.260,00	83
1.	834,25	958,00	87
2.	845,20	958,00	88
3.	831,37	958,00	87
4.	821,10	910,00	90
7.	90,60	93,30	97
8.	0,98	1,06	92
9.	11,19	12,50	90
10.	0,97	1,12	87

Table 5.1: Network bandwidth between nodes.

Chapter 6

Conclusions and Future Work

This document describes two integrated solutions that were devised with a common goal - Easy management and user interconnection across Grid sites -, motivated by the needs of the Distributed Systems projects where DI is involved.

The first solution consists on the EGI roll, which allows easy and efficient configuration, installation and maintenance of Grid sites. It is an evolution from the previous work that was developed by the team[28, 26]. This contribution adapts the roll to recent software releases and architecture changes, while also adding extra functionalities as described in chapter 3.

The Distributed Computing area is evolving at a fast pace, particularly global projects like the EGI. As described in section 2.3.2, there is a plan to migrate the grid middleware from gLite to UMD. This means that all sites should start thinking about that transition. In the future, the roll must be adapted to the new middleware, which should be a fairly simple task, since the architecture remains the same - UMD still follows the same approach, based on appliances with different roles that provide the required services for the Grid ecosystem.

There are other improvements that can be made to the EGI roll. The way the site-wide configuration file is maintained is a clear point that needs further work. The configuration of the Grid site's appliances is done using

YAIM, a tool which takes the *site-info.def* text file, consisting a list of general variables, to configure the different nodes. When doing troubleshooting, the site administrator tends to make changes on this file locally, on the appliance where the problem is occurring, rather than on the central node. This can lead to inconsistency between the different nodes, creating different versions of the file. One solution, and further enhancement to the EGI roll, would be to use a version control system (i.e. subversion) to keep track of those changes and push the modifications to the other nodes of the Grid site.

The second solution is SPM, a tool that enables the interconnection of Grid sites, providing a user-level resource architecture, as part of the User Domains project [1]. To achieve that control over a loosely coupled set of resources, SPM creates a presence and messaging system and builds an efficient communication link between any entities that are successfully registered in that system, including those located behind NAT devices.

SPM delivers a network layer with two distinct functions: i) the gathering of information about live peers and basic session establishment and ii) the direct and reliable interconnection of the peers using NAT traversal techniques.

Security is one subject that must be addressed in the future when thinking about using SPM in a production environment. In that situation, it is critical to harden security both by authenticating the devices that handle the connection and by protecting the exchanged traffic through encryption.

The results in section 5.2.2 show that the use of SPM doesn't inflict a large performance penalty confirming the expectable usability both at high bandwidth and at home computing facilities.

In summary, the integration of the devised solutions achieves scalability and flexibility at multiple levels: a) the EGI roll, to allow the infrastructure to be scaled both site-wide and Grid-wide by simplifying the deployment and administration processes and b) the User Domain abstraction enables a user-level platform, creating an infrastructure overlay that allows users to take full control of their allocated resources leveraging from the computing capacity available in multiple sites.

Bibliography

[1] V. Oliveira, A. Pina, and A. Rocha. Running user-provided virtual machines in batch-oriented computing clusters. In *Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on*, pages 583 –587, feb. 2012. doi: 10.1109/PDP.2012.91.

[2] glitehome. gLite Middleware homepage. `http://glite.web.cern.ch/glite/`.

[3] V. Oliveira, A. Pina, and T. Sa. Simple peer messaging for remote user domains interconnection. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, pages 315 –321, july 2012. doi: 10.1109/HPCSim.2012.6266931.

[4] Philip M. Papadopoulos, Mason J. Katz, and Greg Bruno. Npaci rocks: Tools and techniques for easily deploying manageable linux clusters. *Cluster Computing, IEEE International Conference on*, 0:258, 2001. doi: http://doi.ieeecomputersociety.org/10.1109/CLUSTR.2001.959986.

[5] G. Bruno, M.J. Katz, F.D. Sacerdoti, and P.M. Papadopoulos. Rolls: modifying a standard system installer to support user-customizable cluster frontend appliances. *Cluster Computing, IEEE International Conference on*, 0:421–430, 2004. doi: http://doi.ieeecomputersociety.org/10.1109/CLUSTR.2004.1392641.

[6] Elisa Lanciotti MaartenLitmaath PatriciaMe ndez Lorenzo Vincenzo Miccio Christopher Nater Roberto Santinelli Andrea Sciaba Stephen Burke, Simone Campana. glite 3.2 user guide.

[7] A Maier. Ganga— a job management and optimising tool. *Journal of Physics: Conference Series*, 119(7):072021, 2008. URL http://stacks.iop.org/1742-6596/119/i=7/a=072021.

[8] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), February 1996. URL http://www.ietf.org/rfc/rfc1918.txt.

[9] Rubén Cuevas, Ángel Cuevas, Albert Cabellos-Aparicio, Loránd Jakab, and Carmen Guerrero. A collaborative p2p scheme for nat traversal server discovery based on topological information. *Computer Networks*, 54(12):2071 – 2085, 2010. ISSN 1389-1286. doi: 10.1016/j.comnet.2010.03.022. URL http://www.sciencedirect.com/science/article/pii/S1389128610001817. ¡ce:title¿P2P Technologies for Emerging Wide-Area Collaborative Services and Applications¡/ce:title¿.

[10] P. Srisuresh and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). RFC 3022 (Informational), January 2001. URL http://www.ietf.org/rfc/rfc3022.txt.

[11] G. Tsirtsis and P. Srisuresh. Network Address Translation - Protocol Translation (NAT-PT). RFC 2766 (Historic), February 2000. URL http://www.ietf.org/rfc/rfc2766.txt. Obsoleted by RFC 4966, updated by RFC 3152.

[12] Bryan Ford, Dan Kegel, and Pyda Srisuresh. Peer-to-Peer Communication Across Network Address Translators. In *Proceedings of the 2005 USENIX Technical Conference*, 2005. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.6799.

[13] Nathan S. Evans. Methods for secure decentralized routing in open networks. Master's thesis, Technische Universität München, Garching bei München, 08/2011 2011.

[14] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. URL http://www.ietf.org/rfc/rfc791.txt. Updated by RFC 1349.

[15] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. URL `http://www.ietf.org/rfc/rfc2460.txt`. Updated by RFCs 5095, 5722, 5871.

[16] P. Srisuresh, B. Ford, and D. Kegel. State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs). RFC 5128 (Informational), March 2008. URL `http://www.ietf.org/rfc/rfc5128.txt`.

[17] R. Mahy, P. Matthews, and J. Rosenberg. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 5766 (Proposed Standard), April 2010. URL `http://www.ietf.org/rfc/rfc5766.txt`.

[18] Andreas Müller, Nathan Evans, Christian Grothoff, and Samy Kamkar. Autonomous nat traversal. In *10th IEEE International Conference on Peer-to-Peer Computing (IEEE P2P 2010)*. IEEE, 2010.

[19] G. Huston, A. Lord, and P. Smith. IPv6 Address Prefix Reserved for Documentation. RFC 3849 (Informational), July 2004. URL `http://www.ietf.org/rfc/rfc3849.txt`.

[20] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389 (Proposed Standard), October 2008. URL `http://www.ietf.org/rfc/rfc5389.txt`.

[21] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245 (Proposed Standard), April 2010. URL `http://www.ietf.org/rfc/rfc5245.txt`. Updated by RFC 6336.

[22] Salman A Baset and Henning Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. *Proceedings IEEE INFOCOM 2006 25TH IEEE International Conference on Computer Communications*, 6(c):1–11, 2004. URL `http://arxiv.org/abs/cs/0412017`.

[23] A.S. Ahmed and R.H. Shaon. Evaluation of popular voip services. pages 58 – 63, 2009.

[24] Mikael Goldmann and Gunnar Kreitz. Measurements on the spotify peer-assisted music-on-demand streaming system. *To appear in IEEE P2P'11 (industrial session)*, 2011.

[25] P. Saint-André, K. Smith, and R. Tronçon. *XMPP: the definitive guide : building real-time applications with Jabber technologies*. Definitive Guide Series. O'Reilly, 2009. ISBN 9780596521264. URL `http://books.google.pt/books?id=SG3jayrd41cC`.

[26] Bruno Oliveira, Antonio Pina, and Alberto Proenca. EGEE site administration made easy. In Proenca, A and Pina, A and Tobio, JG and Ribeiro, L, editor, *IBERGRID: 4TH IBERIAN GRID INFRASTRUCTURE CONFERENCE PROCEEDINGS*, pages 295–306, 2010. ISBN 978-84-9745-549-7. 4th Iberian Grid Infrastructure Conference (IBERGRID), Braga, PORTUGAL, MAY 24-27, 2010.

[27] Mason J. Katz, Philip M. Papadopoulos, and Greg Bruno. Leveraging standard core technologies to programmatically build linux cluster appliances. In *CLUSTER 2002: IEEE International Conference on Cluster Computing*, pages 47–53, 2002.

[28] Antonio Pina, Bruno Oliveira, Albano Serrano, and Vitor Oliveira. EGEE Site Deployment & Management Using the Rocks toolkit. In Silva, F and Barreira, G and Ribeiro, L, editor, *IBERGRID: 2ND IBERIAN GRID INFRASTRUCTURE CONFERENCE PROCEEDINGS*, pages 285–295, 2008. ISBN 978-84-9745-288-5. 2nd Iberian Grid Infrastructure Conference (IBERGRID), Porto, PORTUGAL, MAY 12-14, 2008.