



Escola Politècnica Superior
de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

PROYECTO DE FIN DE CARRERA

TÍTULO: Implementation of a Public Key Infrastructure over Peer-to-Peer network

TITULACIÓN: Ingeniería de Telecomunicaciones (segundo ciclo)

AUTOR: Xavi Barrera Quintanilla

DIRECTOR: Sergio Machado Sanchez

FECHA: 18 de Mayo del 2007

Título: Implementación de una Infraestructura de clave pública sobre una red Peer-to-Peer

Autor: Xavi Barrera Quintanilla

Director: Sergio Machado Sanchez

Fecha: 18 de Mayo del 2007

Resumen

En este proyecto se ha desarrollado la aplicación PKI-P2P, esta aplicación implementa una infraestructura de clave pública (PKI) sobre una red peer-to-peer (P2P). Una PKI tiene como objetivo probar que una clave pública es auténtica para un cierto usuario, porque la confianza que se tiene en una clave pública es muy importante para la seguridad en los métodos criptográficos. Lo normal es que el sistema sea centralizado y jerárquico en donde unos pocos elementos llamados Autoridades de Certificación (AC) son los encargados de validar la relación entre un usuario y su clave pública. En redes con una gran cantidad de nodos la PKI tiene que atender muchas peticiones de autenticidad de clave pública, por lo tanto, en este tipo de escenarios es mejor descentralizar la PKI. Para ello todos los elementos de la PKI deberían ser capaces de decidir si una clave pública es auténtica o no. Las redes descentralizadas en donde todos los elementos son iguales son las llamadas P2P, estas redes ofrecen algunas ventajas sobre los sistemas jerárquicos o centralizados como: resistencia a fallos, distribución de carga, auto administración y independencia de organización operativa.

La forma de implementar una PKI sobre una red P2P es descrita en el documento de Thomas Wöfl "Public-Key-Infrastructure Based on a Peer-to-Peer Network", el autor de este documento desarrolló una aplicación Peer-to-Peer-PKI consiguiendo búsqueda y transferencia eficiente de certificados y recomendaciones. Se basa en una combinación del modelo de Maurers y el protocolo escalable de búsqueda P2P de Chord. La red P2P utilizada es Pastry mediante su implementación en java Freepastry, esto ha hecho que todo el proyecto se desarrolle en java. Pastry es un esquema genérico, escalable y eficiente para aplicaciones P2P. Los nodos Pastry forman una red overlay descentralizada, auto-organizada y tolerante a fallos.

Además para probar el funcionamiento de la aplicación PKI-P2P se ha utilizado la red de pruebas PlanetLab. PlanetLab es una red global de investigación para dar soporte al desarrollo de nuevos servicios de red. Gran parte del tiempo se ha dedicado al estudio de PlanetLab, saber como funciona para poder realizar las pruebas.

Title: Implementation of a Public Key Infrastructure over a Peer-to-Peer network

Author: Xavi Barrera Quintanilla

Director: Sergio Machado Sanchez

Date: May, 18th 2007

Overview

In this project has developed the application PKI-P2P, this application implements a Public Key Infrastructure (PKI) on a peer-to-peer (P2P) network. A PKI has as an objective to prove whether a public key is authentic for a certain user, because the confidence that has in a public key is very important for the security in the cryptography methods. The normal thing is that the system be centralized and hierarchical where a few elements called Certification Authorities (CA) are the responsible for validating the relation between a user and their public key. In networks with a large quantity of nodes the PKI has to attend many public key authenticity petitions, therefore, in this type of settings is better to decentralize the PKI. For it all the elements of the PKI should be capable of deciding if a public key is authentic or not. The networks decentralized where all the elements are equals are the P2P networks, these networks offer some advantages on the hierarchical or centralized systems as: fault resistance, load distribution, self administration and independence of an operating organization.

The form to implement a PKI on a network P2P is described in the document of Thomas Wölfel "Public-Key-Infrastructure Based on to Peer-to-Peer Network", the author of this document developed to specialized Peer-to-Peer-PKI realizing efficient search and transfer of certificates and trust-recommendations. It is based on to combination of a logic calculus model for PKIs and a scalable P2P lookup protocol. The network P2P utilized is Pastry by means of its implementation in java Freepastry, this has done that the entire project has been developed in java. Pastry is a generic, scalable and efficient substrate for peer-to-peer applications. Pastry nodes form a decentralized, self-organizing and fault-tolerant overlay network within the Internet.

Besides to test the operation of the application PKI-P2P the PlanetLab network has been used. PlanetLab is a global research network that supports the development of new network services. Great part of the time has been dedicated to study of PlanetLab, to know as functions to be able to carry out the tests.

//dedicatoria

INDEX

INTRODUCTION.....	1
CHAPTER 1. INTRODUCTION TO CRYPTOGRAPHY.....	3
1.1. The cryptography	3
1.2. Cipher Key Algorithms.....	4
1.2.1. Symmetric Key Cipher.....	4
1.2.2. Asymmetric Key Cipher.....	5
1.2.3. Hybrid Systems	6
1.3. Data Integrity.....	7
1.4. Autentication.....	8
1.4.1. Digital Signature	8
1.4.2. Digital Certificates.....	9
1.5. Public Key Infrastructure.....	12
1.5.1. Alternatives.....	14
CHAPTER 2. PASTRY.....	15
2.1. Introduction to the P2P networks	15
2.2. Design of Pastry	15
2.2.1. The nodeId.....	15
2.2.2. State Tables.....	16
2.2.3. Routing algorithm	16
2.2.4. Node arrival	17
2.2.5. Node departure.....	18
2.2.6. Locality.....	19
2.3. FreePastry	21
2.3.1. Introduction	21
2.3.2. Node creation	21
2.3.3. Sending a message.....	21
2.3.4. Implementation of an application on FreePastry	23
2.4. Past	23
2.4.1. Characteristics	23
2.4.2. Insertion of a new element in Past	24
2.4.3. Recover a Past object	24
2.4.4. Replica Diversion.....	24
2.4.5. File Diversion	25
CHAPTER 3. PKI-P2P APPLICATION.....	27
3.1. Introduction.....	27
3.2. Structure and design of the application.....	27
3.2.1. Cryptographic techniques used.....	28
3.3. PKI-P2P elements	30

3.3.1.	User identifier.....	30
3.3.2.	Public messages	31
3.3.3.	Private Statements	32
3.3.4.	Private View and Public View	32
3.3.5.	PAST in PKI-P2P.....	33
3.4.	Trust Model	36
3.4.1.	Insertion of the public key.....	36
3.4.2.	Self-signed certificates and digital signatures	36
3.4.3.	Authentication process	37
3.4.4.	Private messages shipment	40
CHAPTER 4.	PLANETLAB.....	41
4.1.	Introduction.....	41
4.2.	Membership.....	42
4.3.	Elements.....	42
4.4.	Configuration	43
4.4.1.	Use account.....	43
4.4.2.	Slice expiration	45
4.4.3.	Nodes management	46
4.4.4.	Nodes access	47
4.5.	PlanetLab tests	48
4.5.1.	Vxargs: parallel ssh access	51
4.5.2.	Start application.....	51
4.5.3.	Logs.....	54
CONCLUSIONS.....		55
Future works.....		56
Environmental Impact.....		57
Personal Conclusions.....		57
REFERENCES.....		59
ACRONYMS		61
ANNEX A. TEST WITH 50 NODES.....		63
ANNEX B. PLANETLAB TOOLS.....		64
A.1 Introduction.....		64
B.1 CoDeeN.....		64
A.	CoBlitz	65
B.	CoDeploy	66
C.	CoDNS.....	66
D.	CoTop.....	66
E.	CoMon	66

F.	CoTest	67
G.	CoViz	67

ANNEX C.	SCRIPTS	69
-----------------	----------------------	-----------

ANNEX D.	LOG4JAVA CONFIGURATON FILE.....	73
-----------------	--	-----------

INDEX OF FIGURES

Fig 1.1 Symmetric Key Cipher.....	5
Fig 1.2 Asymmetric Key Cipher	6
Fig 1.3 hybrid system	7
Fig 1.4 Data Integrity	7
Fig 1.5 Suplantación de identidad	9
Fig 1.6 Digital Signature	9
Fig 1.7 Certificate example.....	11
Fig 1.8 Chain of certificates	13
Fig 1.9 Public Key Infrastructure	14
Fig 2.1 Routing a message with key=1225.....	16
Fig 2.2 Node arrival.....	18
Fig 2.3 Routing step distance.	20
Fig 2.4 Sending directly a JoinRequest message from A to B.....	22
Fig 2.5 Route Message	22
Fig 3.1 Layers.....	27
Fig 3.2 SubjectPublicKeyInfo syntax	28
Fig 3.3 PrivateKeyInfo syntax.....	29
Fig 3.4 PKICipher class.....	30
Fig 3.5 Cert.....	31
Fig 3.6 Rec	31
Fig 3.7 Message Token.....	32
Fig 3.8 Private and Public View.....	33
Fig 3.9 SecretKey and Index k generation for a certificate	34
Fig 3.10 SecretjKey and Index K generation for a recommendation	34
Fig 3.11 Chain of certificates.....	37
Fig 3.12 Process to obtain the public key of Bob.....	37
Fig 3.13 Authentication process	38
Fig 4.1 Institutions and industrial reserach labs.....	41
Fig 4.2 Form for user account	43
Fig 4.3 PlanetLab user menu	44
Fig 4.4 User account information.....	45
Fig 4.5 Add nodes in PlanetLab	46
Fig 4.6 All nodes in PlanetLab.....	47
Fig 4.7 SSH public key	48
Fig 4.8 Bootstrap output terminal	52
Fig 4.9 vxargs output.....	53
Fig 4.10 Result folder	53
Fig 0.1 CPU Slice visualization.....	68

INTRODUCTION

Currently Internet is being increasingly more popular; the new technologies cover each time more homes taking each time more importance in our lives. The success of Internet is owed in part to that the TCP/IP protocol is free, there is no proprietary of Internet, there is not any central authority, and any person can be connected to Internet. This facility of access is the main attraction since the commercial point of view but is also the cause that Internet be open to all kinds of threats. It is not easy to intercept a communication through Internet but is possible.

On the other hand techniques for the protection of the data in electronic communications exist like for example the data encryption so that only the recipient of those data can understand them. Other techniques are the certificates and the digital signature that guarantee the identity of the subjects. The technology PKI (Public Key Infrastructure) is a combination of all these cryptographic technical that permit to the users to be authenticated, to utilize certificates, to cipher, to decipher, to sign digitally information, to guarantee the not repudiation,...

The main theme of this work is to apply the security of a Public Key Infrastructure to a Peer-To-Peer (P2P) network. The objective is to create an application in java [1] that provides all the functionalities of a PKI and implements on the Pastry network. The Pastry network already is implemented in the FreePastry package for which all the efforts have been centered in implementing the PKI. Therefore the project has followed the document "Public-Key-Infrastructure based on to Peer-to-Peer Network" of Thomas Wölfel [2], in this document is explained how to integrate a PKI in a P2P network. Besides implementing the PKI on Pastry, due to that a P2P network is composed of various nodes, the tests of the application have done them on the PlanetLab test network. Therefore a great part of the project has consisted of studying and to evaluate this network of tests.

The report is comprised of 4 chapters that describe so much the part of the project related to the PKI as the part of study of the PlanetLab. In the chapter 1 an introduction to cryptography is done explaining the cipher algorithms, the digital certificate, the digital signature, etc... In the chapter 2 is entered in the explanation of the Pastry network, in which consists, which are its characteristics, which elements have, etc. Besides it is explained that is FreePastry, and the DHT Past. In the chapter 3 is spoken in detail on the PKI-P2P application that is implemented in this project, which are its functionalities, which is the procedure of authentication, that elements intervene, etc. To finish the chapter 4 tries to do an approximation to the PlanetLab network explaining in which consists, how can be utilized, how are the nodes, the access to them, how to upload files ... creating thus a user manual.

CHAPTER 1. Introduction to cryptography

1.1. The cryptography

The cryptography is the art or science to cipher information by means of mathematical techniques, so that the result is only legible for the receiver. Its purpose is to offer:

- **Confidentiality:** it guarantees that information is only legible for the parts authorized to see it. This is obtained with the cipher of information by means of cipher algorithms.
- **Integrity:** it guarantees that information has not been altered during the transmission of this.
- **Authentication:** it implies to prove the identity of a part to the other part.
- **Non-Repudiation:** the sender can't deny the emission of a message.
- **No-Forward:** it implies that an information or message can't be forwarded by which has captured a legal transaction.

Its history is very extensive [3], the cryptographic techniques already were used at the time of the first Egyptian, Hebrew and Babylonian. The first cryptographic system, denominated "the Scitala", was used by the Spartans in 400a.C and consisted of altering the order of the signs of a text. Its use was military, the messages in "the Scitala" were written on a fabric that wraps up a stick. The message only could be read when it was coiled on a stick of same thickness. Another one of the first methods was the created one by Julio Cesar, based on the substitution of each letter by which it beyond occupies three positions in the alphabet.

The classic methods of the cryptography are not infallible; the message can be obtained because sometimes the algorithm can be broken with a simple calculation. At the present time, the new technologies have allowed to create cryptographic systems more trustworthy and more complexes.

Its use has been and is very extensive in the military environment, but due to the expansion of the network, the security also happens to be important in the communications, due to the great amount of attacks and the different mechanisms to alter the information of transit in the network.

A useful categorization of these attacks is in terms of passive attacks and active attacks. Passive attacks are in the nature of monitoring of transmissions. The goal of the attacker is to obtain information that is being transmitted. Two types of passive attacks are release of message content; traffic analysis. A release of message content is easily understood. A telephone conversation, an electronic mail message, and a transferred file may contain sensitive or confidential information.

The second passive attack, traffic analysis, is more subtle. Suppose that we had a way of masking the contents of a message or other information traffic so that Cuba, even if they capture the information, could not extract the real information because of the use of encryption. The attacker could after a period of time extract the information and messages, defeating the encryption process.

The second major category of attacks is active attacks. These attacks involve some modification of the data stream or the creation of a false stream. It can be subdivided into four categories: masquerade, replay, modification of message, and denial of service.

A masquerade takes place when the attacker, under certain entity, pretends to be a different entity, and therefore enabling an authorized entity to obtain extra privileges. Replay involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect.

Modification of service simply means that some portion of a legitimate message is altered, or that messages are delayed or reordered, to produce an unauthorized effect. The denial of service prevents or inhibits the normal use or management of communications facilities. This is a very important and serious possible attack. It could disrupt an entire network, either by disabling the network or by overloading it with messages so as to degrade performance. The attacker could target airports, financial centers, power companies, dams control centers, etc. It is quite difficult to prevent active attacks. The goal is to detect them and to recover from any disruption or delays caused by them.

Nowadays the cryptography tries to defend against any attack guaranteeing a safe communication to us in networks even opened as it can be Internet.

1.2. Cipher Key Algorithms

A cipher algorithm [5] is a mathematical formula designed specifically to obscure the value and content of data. Most valuable cipher algorithms use a key as part of the formula. This key is used to encrypt the data and either that key or a complementary key is needed to decrypt the data back to a useful form. There are two types of ciphers: the symmetric key cipher and the asymmetric key cipher.

1.2.1. Symmetric Key Cipher

A symmetric key cipher uses the same key to encrypt and decrypt data (see Fig 1.1). Therefore the encrypt algorithm and the decrypt algorithm are complementary and this fact entails to that the sender and receiver must put in agreement in a common secret key.

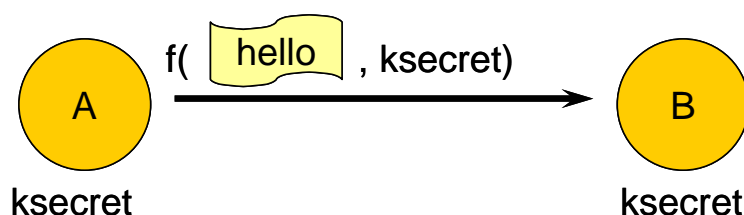


Fig 1.1 Symmetric Key Cipher

Nowadays, the problem is that the computers can guess a key quickly; this determines that the size of the secret key is important. Many cipher algorithms increase their protection by increasing the size of the keys they use. However, the larger the key, the more computing time is needed to encrypt and decrypt data. So, choosing an appropriate cipher algorithm that strikes a balance between your protection needs and the computational cost of protecting that data is important. For example, the algorithm DES uses a key of 56 bits which implies 72.057.594.037.927.936 possible keys. A conventional computer would take days in guessing a key; a specialized computer can take hours. The most recent algorithms like 3DES, Blowfish and IDEA use keys of 128bits, and with this length already it is considered improbable that a machine can guess a key.

The easy implementation of these algorithms as well as the low computational cost for encrypt of a text are an important advantages. In the other hand the secret key has to be known by two points of communication, this is a serious drawback because the secret key has to be share by a secure channel. The security of theses algorithms is in secret key, the attacker prefers to try to intercept the secret key instead of guessing it. A solution of this problem is the use of asymmetric cipher key algorithm.

1.2.2. Asymmetric Key Cipher

The cryptographic algorithms based on asymmetric key cipher use a pair of keys instead of a single key like the symmetric key ciphers. One of these keys is shared with the rest of users, it's a public key, and anybody can have knowledge it. And the other is a private key, this implies that only the owner must know it and must keep in a safe form.

An asymmetric key cipher uses two separate but related keys; one is used to encrypt the data while the other is used to decrypt the data (see Fig 1.2)

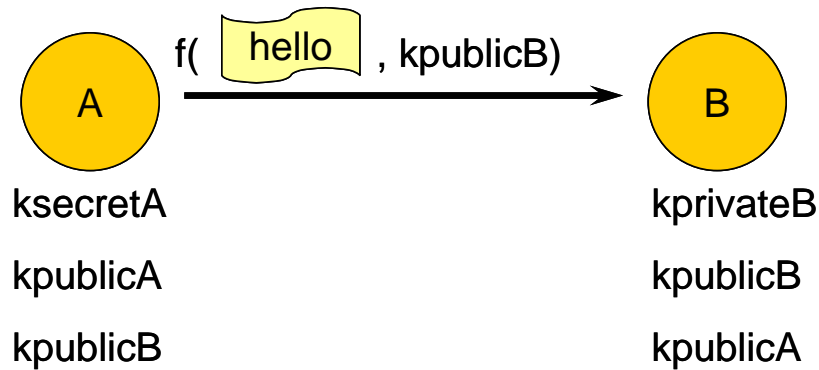


Fig 1.2 Asymmetric Key Cipher

Node A wants to send an encrypted message to node B, but node A needs the node B public key. The node A ciphers the message with de node B public key, therefore the node B is the one able to decrypt the message by means of the private key. The use of a pair of keys avoids the dangerous exchange of a secret key; this is the principal purpose of this type of algorithms. The only requirement of the asymmetric key algorithm is that the sender can obtain the receiver public key.

One of the important characteristics of these algorithms is that they are based on one-way functions, these functions have an easy computation but its inverse function is extremely difficult. But its functions with a trap, this trap allow making its inverse of simple way. In the algorithms of asymmetric key the trap is the private key. In the same way which with the symmetrical key algorithm, the security is in the keys, which implies that their length is important. In the case of the asymmetric key algorithms the use of keys of 1024 bits is recommended, because these algorithms are based on factorization of prime numbers

In these algorithms it doesn't exist a secret key to share with the rest of nodes, this is the great advantage of this system, but the drawback is that the computational time is increased, furthermore the keys are more larger as well as the encrypted messages.

Examples of asymmetric key algorithms are RSA and DSA.

1.2.3. Hybrid Systems

Some systems use both algorithms, symmetric and asymmetric, taking advantage of the advantages that offer types both. A secret key is generated, and the asymmetric key algorithm is used to share that secret key, leaving the transmission of the information for the algorithm of symmetric key algorithm.

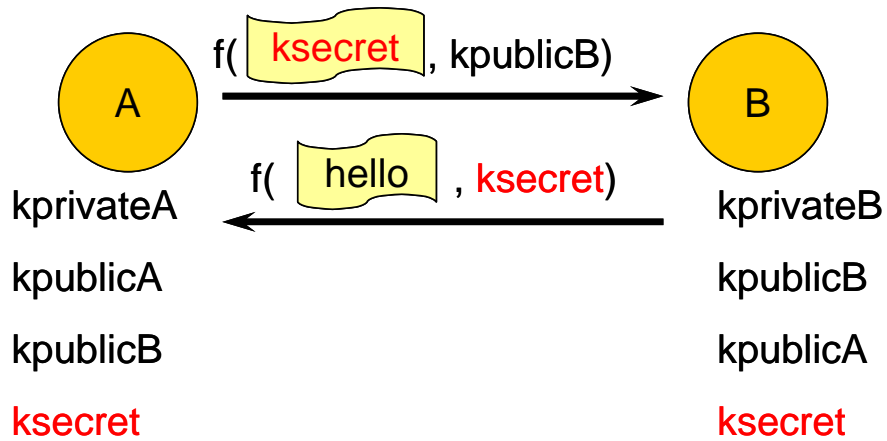


Fig 1.3 hybrid system

Node A generate the secret key and encrypt it by means of the node B public key, therefore the node B is the one able to decrypt the secret key (see Fig. 1.3).

Some tools like PGP or SSH using this type of hybrid systems.

1.3. Data Integrity

The integrity of the data is obtained applying Hash functions to a message. A Hash function generates a cryptographic checksum from a text, all checksums generates by Hash functions has the same size, therefore that applying to two equal texts a Hash function will obtain two identical checksums, this property prevent to find out some characteristic of the original text. In addition the checksums are uniques, so that to obtain the same checksum it's necessary to apply the Hash function to the same input. Hash functions are also called one-way functions because it is easy to determine the hash from the message but mathematically infeasible to determine the message from the hash.

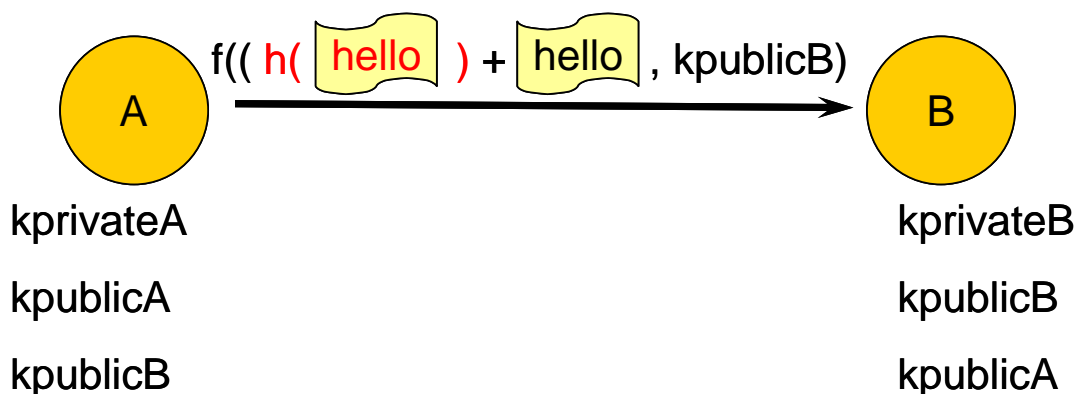


Fig 1.4 Data Integrity

In order to obtain integrity in the transmission of a message it's necessary to apply a Hash function to the original message and encrypt the checksum with the message and send it (see Fig. 1.4). The receiver decrypts the message and it applied the same Hash function to the original message, and the new checksum is generated. Next, the receiver has to compare both checksums, if both are equal means that the sender is the one who claims to be.

For integrity, you can choose between two hash functions when setting policy:

- MD5: Message Digest 5 (MD5) is based on RFC 1321. MD5 completes four passes over the data blocks, using a different numeric constant for each word in the message on each pass. The number of 32-bit constants used during the MD5 computation ultimately produces a 128-bit hash that is used for the integrity check.
- SHA1: Secure Hash Algorithm 1 (SHA1) was developed by the National Institute of Standards and Technology as described in Federal Information Processing Standard (FIPS) PUB 180-1. The SHA process is closely modeled after MD5. The SHA1 computation results in a 160-bit hash that is used for the integrity check. Because longer hash lengths provide greater security, SHA is stronger than MD5.

1.4. Authentication

Could it be that somebody intercept a secure communication and injects own messages in order to the receiver thinks that these messages are of the legitimate sender. This case creates a necessity of authentication of the sender.

One of the intentions of an authentication scheme is to detect if somebody has modified the original message. For example an application would be the authentication in the distribution of the public keys. The distribution of the public key is difficult, because we cannot assure that if I obtain a public key its owner is the real owner, therefore an authentication key system is necessary.

In order to authenticate a message the digital signatures are the better solution, these signs are able to assure if a message comes from a sender or another sender.

1.4.1. Digital Signature

In picture 1.5 the node A send an authentication message to node B, if node C intercepts this authentication message and repeat it, the node C could to replace the node A identity, although the node A include a password in the message. Anything that is to send a text causes that the authentication fails.

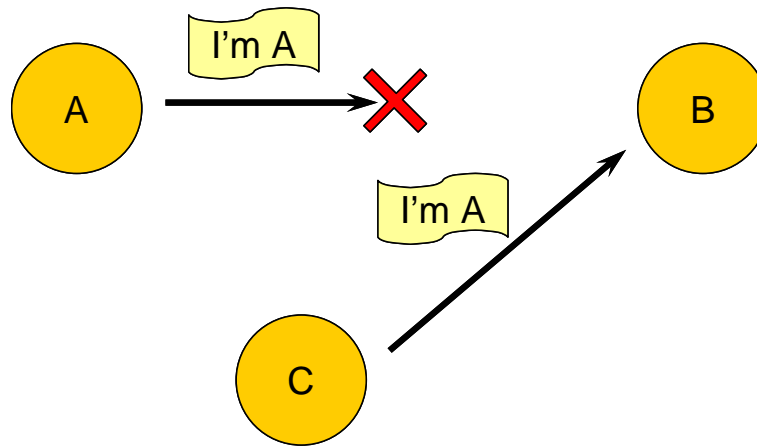


Fig 1.5 Suplantación de identidad

A solution is to verify that node A has a private key (because the node A is the only that has the private key). The procedure would be: The node B sends a code to node A (see Fig. 1.6). The node A encrypts this code by means of its private key, the result is sent to node B. The node B decrypts the message with the node A public key (the node A public key is shared with the rest of nodes) and verifies if the code is the original code that the node B sent at first. So that the node A it demonstrates to node B that really is the node A.

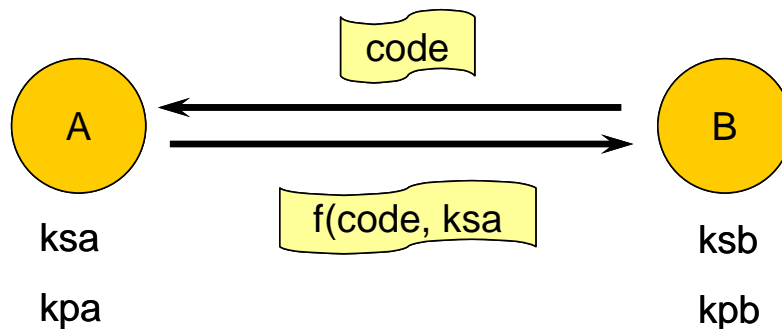


Fig 1.6 Digital Signature

In summary, the signature process consists in apply the Assymmetric key algorithm the other way around, encrypts the text with the private key and decrypts with public key.

1.4.2. Digital Certificates

Normally the digital certificates are used to authenticate a public key. These certificates contain the public key besides several information on the owner of the key, and the most important thing, it is that a digital signature of a third party entity includes. This third party entity is named Certificate Authorities (CA), and

also attests that the public key contained in the certificate belongs to the person, organization, server or other entity noted in the certificate. A CA's obligation in such schemes is to verify an applicant's credentials, so that users and relying parties can trust the information in the CA's certificates. If the user trusts the CA and can verify the CA's signature, then they can also verify that a certain public key does indeed belong to whomever is identified in the certificate. If the CA can be subverted, then the security of the entire system is lost.

To obtain a digital certificate you must direct to CA and give the public key and your personal information, the CA commission of creating the certificate and signing it. There is assumed CA's hierarchic strict system for the issue of the certificates, so that only a CA can emit certificates. There exist several formats of digital certificate, but more commonly used it is the marked for the standard X-509v3 [6].

X.509 was initially issued in 1988 and was begun in association with the X.500 standard and assumed a strict hierarchical system of certificate authorities (CAs) for issuing the certificates. This contrasts with web of trust models, like PGP, where anyone (not just special CAs) may sign, and thus attest to the validity of others' key certificates. Version 3 of X.509 includes the flexibility to support other topologies like bridges and meshes. It can be used in a peer-to-peer, OpenPGP-like web of trust. The X.500 system has never been fully implemented, and the IETF's Public-Key Infrastructure (X.509), or PKIX, working group has adapted the standard to the more flexible organization of the Internet. In fact, the term X.509 certificate usually refers to the IETF's PKIX Certificate and CRL Profile of the X.509 v3 certificate standard, as specified in RFC 3280, commonly referred to as PKIX for Public Key Infrastructure (X.509).

A certificate contains normally the name of the certified entity, personal information (name, IP, DNI), a number of series, the expiry and creation date, a copy of the public key, information of the CA, the CA's identifying and the digital signature of the CA. Then there comes a part that contains the cipher algorithms and Hash's functions that have to be in use for encrypt process.

La sintaxis del certificado se define utilizando ASN.1 (Abstract Syntax Notation One). A continuación se muestra un ejemplo de certificado X.509 de www.freesoft.org firmado por la empresa Thawte.

The syntax of the certificate is defined using ASN.1 (Abstract Syntax Notation One). Later there appears an example of certificate X.509 of www.freesoft.org signed by the Thawte Company.

```

Certificate:
Data:
  Version: 1 (0x0)
  Serial Number: 7829 (0x1e95)
  Signature Algorithm: md5WithRSAEncryption
  Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte
  Consulting cc,
    OU=Certification Services Division,
    CN=Thawte Server CA/Email=server-certs@thawte.com
  Validity
    Not Before: Jul 9 16:04:02 1998 GMT
    Not After : Jul 9 16:04:02 1999 GMT
  Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
    OU=FreeSoft, CN=www.freesoft.org/Email=baccala@freesoft.org
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (1024 bit)
      Modulus (1024 bit):
        00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
        33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
        66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
        70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
        16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
        c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
        8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
        d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
        e8:35:1c:9e:27:52:7e:41:8f
      Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
    93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
    92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
    ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
    d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
    0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
    5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
    8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
    68:9f

```

Fig 1.7 Certificate example

It was issued by Thawte (since acquired by Verisign), as stated in its Issuer field. Its subject contains a lot of personal information, but the most important part is the common name (CN) of `www.freesoft.org` - this is the part that must match the host being authenticated. Next comes an RSA public key (modulus and public exponent), followed by the signature, computed by taking an MD5 hash of the first part of the certificate and encrypting it with Thawte's RSA private key

There is a special case of certificates, when the certificate is not signed by any CA but by the owner of the public key. In this case we are speaking about **self-signed certificates**.

1.5. Public Key Infrastructure

A Public Key Infrastructure (PKI) [4] is a combination of hardware and software, political and safety procedures that allow the execution of electronic transactions with guarantees of cryptographic operations as the cipher, the digital signature or non-repudiation.

Therefore in a PKI there will be had cipher algorithms, digital certificates, CAs, Hash's functions and digital signatures. All this set of technologies based on public key cryptography allows a user: to be authenticated opposite to the others, non-repudiation (to prevent that once signed a document the signer retracts or denies to have written it), the integrity of the information (to prevent the deliberate or accidental modification of the signed information, during its transport, storage or manipulation), and the agreement of secret keys to guarantee the confidentiality of the interchanged information, it is signed or not.

The most important functionality is the authentication in order to provide a proof of authenticity of the owner of a public key. For example, to a user A there comes a public key of the user B, and user A throws a request to the PKI on this user B public key. The PKI will provide proofs of authenticity to him to know if the user B is the owner of the public key or not, this is done using digital signed messages for the third user C. These messages are the digital certificates, which contain the public key of the user B and are signed by a third entity, a CA. But with the certificate only it is not sufficient, has to pass three proofs:

- Is the public key of the third entity authentic? It is necessary to verify the digital signature.
- The certificate is valid or has expired?
- The user A trusts in the third entity.

To verify the digital signature can take us to another certificate, for example a user C has signed the certificate, for what it is necessary to verify that the public key of user C is really of user C. So that the second certificate is obtained, in this new certificate another entity has signed, therefore also it is necessary to apply the three proofs before mentioned. The result is that chain of certificates is generated, for example some CAs are signed to others forming a chain of certificates (see Fig. 1.8). These chains begin in a certificate called Trust Anchor in which the user A trusts fully.

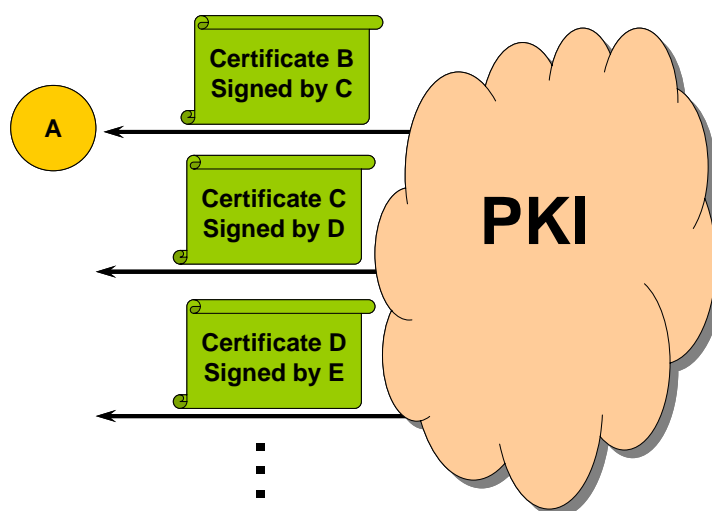


Fig 1.8 Chain of certificates

In a PKI the cipher algorithms are the same and are known by all users, so that the safety relapses into the private key of every user, therefore its must be stored in an as sure as possible way. In addition, all the certificates of a PKI only can be emitted by a recognized Certification Authority. Another important aspect of a PKI is that does not need of the interchange of any type of secret key in the establishment of a PKI safely communications, for which they are considered to be very sure.

The habitual components in a PKI are:

- Certification Authority: it emits and revokes the certificates and the whole confidence of the certificates relapses into it.
- Repositories: it is where there is stored all the information relative to the PKI. The most important are the list of revoked certificates, which is where the not valid certificates are included, and the repository of certificates in which the valid certificates are included.
- Users or final entities: they possess one par of keys, public and private, and a certificate associated with the public key.
- The risk authority: its manager of verifying the link between certificates and the identity of his holders.
- The validation authority: it takes charge verifying the validity of the certificates

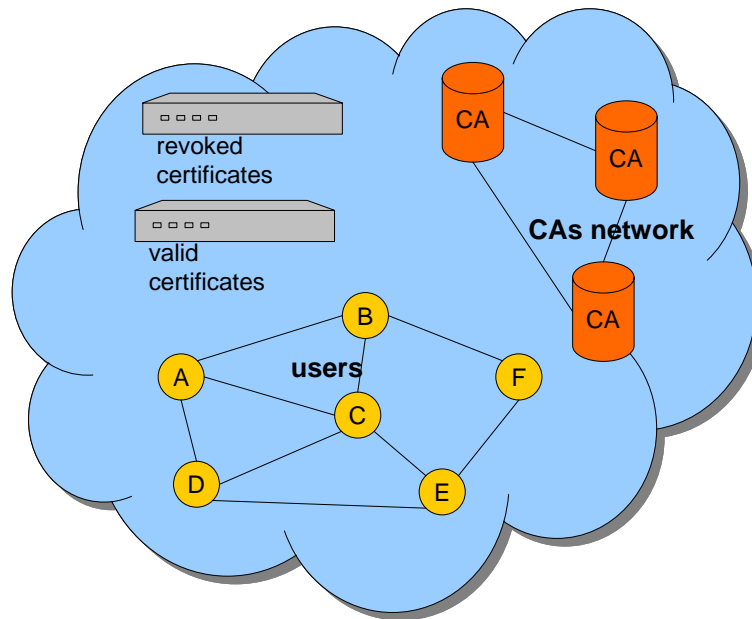


Fig 1.9 Public Key Infrastructure

Nowadays, a PKI is very useful in nets as Internet since it is the only way of giving confidence to the actors of the telematic relations, so much in the business-to-business between companies, as in the trade to for minor, between selling and Internet buying individuals.

The key is also in the confidence in a group of CAs throughout the world recognized (as VeriSign) or locally accepted (as Camerfirma, ipsCA, FNMT, ACE or FESTE in Spain) it is allowed that the involved entities could rely some of others, neither in spite of physical contact nor previous link exist between the parts.

1.5.1. Alternatives

An alternative approach to the problem of authentication of public key information across time and space is the web of trust scheme, which uses self-signed certificates and third party attestations of those certificates. Examples of implementations of this approach are PGP (Pretty Good Privacy) and GnuPG (The GNU Privacy Guard; a free implementation of OpenPGP, the standardized specification of PGP). Because of PGP's (and clones) extensive use in email, the Web of Trust originally implemented by PGP is the most widely deployed bidirectional PKI.

A newer and rapidly growing alternative is the simple public key infrastructure (SPKI) that grew out of 3 independent efforts to overcome the complexities of X.509 and the anarchy of PGP's web of trust. SPKI binds people/systems directly to keys using a local trust model, similar to PGP's web of trust, with the addition of authorization integral to its design.

CHAPTER 2. Pastry

2.1. Introduction to the P2P networks

During the last years, applications like Napster, Gnutella or FreeNet have popularized the P2P applications so much by the polemics by themes of copyright as by the different interesting aspects that contribute. Some of these aspects they are: decentralized control, self-organization, adaptation and scalability. Nevertheless one of the big problems that have the programmers of applications P2P is that of creating algorithms of routing and locating of objects that be efficient.

Pastry [7], a generic peer-to-peer object location and routing scheme, based on a self-organizing overlay network of nodes connected to the Internet. Pastry is completely decentralized, fault-resilient, scalable, and reliable. Moreover, Pastry has good route locality properties. Pastry is intended as general substrate for the construction of a variety of peer-to-peer Internet applications like global file sharing, file storage, group communication and naming systems.

Each node in Pastry has a unique identifier; this identifier is called *nodeld*. The main function of Pastry consists of, given a message with a key, a Pastry node efficiently routes the message to the node with a *nodeld* that is numerically closest to the key, among all currently live Pastry nodes. Therefore, we can say that, a Pastry system is a overlay network of nodes self-organized, where each node routes petitions of clients and relates to an or more local applications.

2.2. Design of Pastry

2.2.1. The *nodeld*

Each node in the Pastry peer-to-peer overlay network is assigned a 128-bit node identifier (*nodeld*). The *nodeld* is used to indicate a node's position in a circular *nodeld* space, which ranges from 0 to $2^{128}-1$. The *nodeld* is assigned randomly when a node joins the system. It is assumed that *nodelds* are generated such that the resulting set of *nodelds* is uniformly distributed in the 128-bit *nodeld* space. For instance, *nodelds* could be generated by computing a cryptographic hash of the node's public key or its IP address. Assuming a network consisting of N nodes, Pastry can route to the numerically closest node to a given key in less than $\log_2^b N$ steps under normal operation (b is a configuration parameter with typical value 4).

2.2.2. State Tables

To send a message to a node does lack to have knowledge of some of the nodes that conform the network Pastry. For it two structures of data are created: the routing table and the leafset. A node's routing table, R , is organized into $\lceil \log_2 bN \rceil$ rows with $2^b - 1$ entries each. The $2^b - 1$ entries at row n of the routing table each refer to a node whose nodeId shares the present node's nodeId in the first n digits, but whose $n+1$ th digit has one of the $2^b - 1$ possible values other than the $n+1$ th digit in the present node's id.

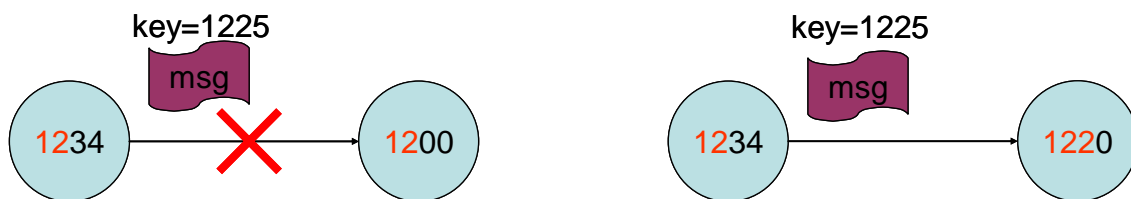


Fig 2.1 Routing a message with key=1225.

In the figure 2.1 a graphic example can be observed of what has been described further up. The node with nodeId 1234 should seek in its state tables a node that share with the key a prefix that be a more long digit than the one that shares, or that be more close to the key that it. The node with nodeId 1234 shares with the key the two first digits (12).

On the other hand, the leaf set L is the set of nodes numerically closest to local node. This set is divided in two parts: the $|L|/2$ numerically closest larger nodeIds , and the $|L|/2$ nodes with numerically closest smaller nodeIds , relative to the present node's nodeId . The leaf set is used during the message routing, as described below. Typical values for $|L|$ are 2^b or 2×2^b .

2.2.3. Routing algorithm

A Pastry node is able to route a message in an efficient way by means of the state tables and the routing algorithm.

Next, the Pastry routing procedure is shown in pseudo code form:

$R \Rightarrow$ Routing table, $L \Rightarrow$ leafset, $D \Rightarrow$ key, $A \Rightarrow$ local nodeId

```

if ( $D$  is within range of our leafset){
    Forward to  $L_i$ , where  $L_i$  is the  $\text{nodeId}$  in leafset more numerically closest
    to  $D$ .
} else {
    Use the routing table.

```

We seek a node that shares a common prefix with the D by at least one more digit.
if (the entry is doesn't empty){
 Forward message to the node of the entry.
} else {
 This is a rare case. The message is forwarded to a node that shares a prefix with the D at least as long as the A, and is numerically closer to the D than the present node's id.

This simple routing procedure always converges, because each step takes the message to a node that either shares a longer prefix with the key than the local node, or shares as long a prefix with, but is numerically closer to the key than the local node.

Three cases in the routing procedure can be differentiated:

- If the key is within range of the leafset, then the destination node is at most one hop away.
- If a message is forwarded using the routing table, then the set of nodes whose nodelds have a longer prefix match with the key is reduced by a factor 2^b in each step, which means the destination is reached in $\log_2^b N$ steps.
- If the key is not covered by the leafset, but there is no routing table entry, and assuming no recent failures, this means that a node with the appropriate prefix does not exist.

The likelihood of this case given the uniform distribution of nodelds, depends on L. If $L = 2^b$ the probability of that this arise is smaller that 0'02, and if $L = 2 \times 2^b$, the probability is smaller that 0'006 (see [1]). Nevertheless, when it happens, no more than one additional routing step results with high probability.

In the event of many simultaneous node failures, the number of routing steps required may be at worst linear in N, while the nodes are updating their state. In this case, eventual message delivery is guaranteed unless L/2 nodes with consecutive nodelds fail simultaneously.

2.2.4. Node arrival

When a new node arrives, it needs to initialize its state tables, and then inform other nodes of its presence. We assume the new node knows initially about a nearby Pastry node A (known as the bootstrap node), according to the proximity metric, that is already part of the system. Such a node can be located automatically, for instance, using "expanding ring" IP multicast, or be obtained by the system administrator through outside channels. Let us assume the new node's nodeld is X. First, the node X routes a join message with key equal to X, but the node X can't to route a message because it isn't still a Pastry node. Therefore, the node X asks A to route the message. Like any message, Pastry routes the join message to the existing node Z whose nodelds is numerically closest to node X.

In response to receiving the join request, nodes A, Z, and all nodes encountered on the path from A to Z send their state tables (see fig. 2.2).

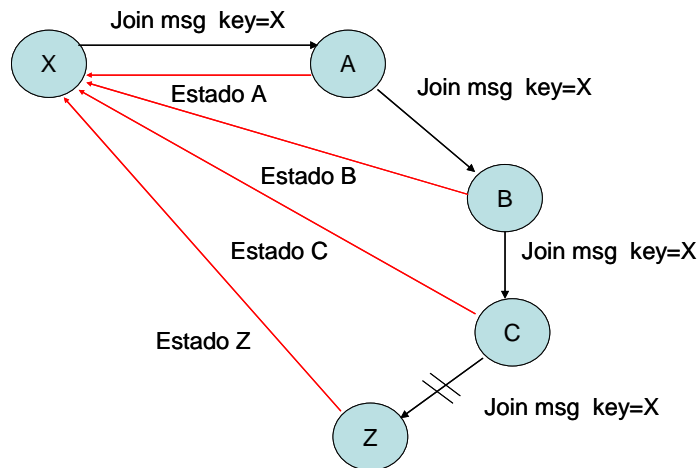


Fig 2.2 Node arrival

The new node X inspects this information, may request state from additional nodes, and then initializes its own state tables. First, to initialize the leafset, X utilizes as base the leafset of Z, since Z is the node numerically more close to X.

To initialize the routing table, X uses all the routing tables that have sent him. For the row 0 of the routing table (X_0), X utilizes the row 0 from A (A_0), because A and X they share no prefix, therefore A_0 is adequate for X_0 . For the row 1 (X_1), the appropriate values would be those of B1, because A and B already they share the first digit of their nodeid. Similarly, X obtains appropriate entries for X_2 from node C, the next node encountered along the route from A to Z, and so on. If A and X to share prefix would not pass anything, for X_0 would continue being utilized A_0 , but where leaves A would be X.

Finally, X transmits a copy of its resulting state to each of the nodes found in its leaf set, and routing table. Those nodes in turn update their own state based on the information received. One can show that at this stage, the new node X is able to route and receive messages, and participate in the Pastry network. The total cost for a node join, in terms of the number of messages exchanged, is $O(\log_2^b N)$.

2.2.5. Node departure

A Pastry node is considered failed when its immediate neighbors in the nodeid space can no longer communicate with the node. When this occurs every node that contain the failed node in its leafset or its routing table should be brought update. For instance, if a node X fails and there is a node A that has to X in its leafset (L), A should eliminate it. To replace a failed node in the leaf set, its

neighbor in the nodeid space contacts the live node with the largest index on the side of the failed node, and asks that node for its leafset. This procedure guarantees that each node lazily repairs its leafset unless $L/2$ nodes with adjacent nodeids have failed simultaneously. Due to the diversity of nodes with adjacent nodeids, such a failure is very unlikely even for modest values of L .

The failure of a node that appears in the routing table of another node is detected when that node attempts to contact the failed node and there is no response. If the failed node is found in the routing table is not very serious but must repair it: if the failed entry is the number d of the row l (R_l^d), then we contact with another node of the same row l , for example the number i (R_l^i). To this node R_l^i , we ask him on the node that has in the entry R_l^d of his routing table. The local node uses the node that have in that entry to repair his entry (before the node looks if is alive). If no node of the row l can be contacted, tries with the nodes of the row $l + 1$. This procedure is highly likely to eventually find an appropriate node if one exists.

2.2.6. Locality

Pastry routes always to the closest node numerically in an efficient way, but besides, the route chosen for a message is likely to be “good” with respect to the proximity metric. Pastry’s notion of network proximity is based on a scalar proximity metric, such as the number of IP routing hops or geographic distance. It is assumed that the application provides a function that allows each Pastry node to determine the “distance” of a node with a given IP address to itself. A node with a lower distance value is assumed to be more desirable.

Propiety 1: All routing table entries refer to a node that is near the present node, according to the proximity metric. If it recalls the example of the previous section, the new node X asks an existing node A to route a join message using X as the key. The message follows a path through the nodes $A, B, C...$ until arriving at Z , which is the alive node with the numerically closest nodeid to X . It is assumed that A is a closest node (in metric of proximity) to X ; also is assumed that this property was maintained before X did join. If this complies, the entries of the row 0 of A (A_0), they are close to A , and since A is close to X , A_0 will be nearby also to X . For the time being the property is maintained. For the row 1 of X (X_1) is caught the row 1 of B (B_1); these entries are close to B , but does not seem that they be going to be near X since is not known how closely is B of X . In reality the entries tend to be reasonably close to X , because the distance that there is of B to B_1 is larger than that of B to X . As a consequence we have that B_1 is a good option for X_1 . This argument applies to the following levels of the board of route

After X has initialized its state in this fashion, its routing table approximate the desired locality property. However, the quality of this approximation must be improved to avoid cascading errors that could eventually lead to poor route locality. For this purpose, there is a second stage in which X requests the state from each of the nodes in its routing table. It then compares the distance of

corresponding entries found in those nodes' routing tables, respectively, and updates its own state with any closer nodes it finds.

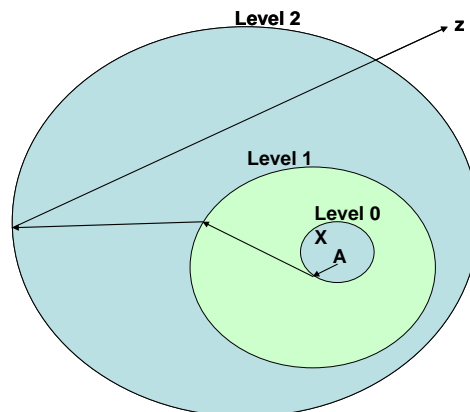


Fig 2.3 Routing step distance.

In the figure 2.3, a representation of the routing of the join message from A to Z can be seen. Each step travels through an upper distance and this does that to each hop we approach more quickly to node Z. Each circle represents the average distances to which are the entries of a certain row of each node that go finding in the path. For example, the level0 circle, represents the entries of the row 0 of the node A; the level1 circle, represents the entries of the row 1 of the node B; and so on. Observing well the figure can be seen that X always is inside the circles, what means that all those levels are adequate for their routing table. Nevertheless X does not remain in the center of the circles; this gives rise to the phase 2 described previously, where the node X asks the state of each entry that discovers in the phase 1.

Experimental results in [1] show that this procedure maintains the locality property in the routing table and neighborhood sets with high fidelity.

Propiety 2: The entries in the routing table of each Pastry node are chosen to be close to the present node, according to the proximity metric, among all nodes with the desired nodeld prefix. As a result, in each routing step, a message is forwarded to a relatively close node with a nodeld that shares a longer common prefix or is numerically closer to the key than the local node. That is, each step moves the message closer to the destination in the nodeld space, while traveling the least possible distance in the proximity space.

Besides, like Pastry only uses local information in the routing, causes that the next hop in the route be chosen without a global sense of direction. This procedure clearly does not guarantee that the shortest path from source to destination is chosen; however, it does give rise to relatively good routes.

Propiety 3: Among a set of k nodes with nodelds nearby numerically to the key, Pastry tends to route toward the most nearby one in terms of metric of proximity

Some P2P applications we have built using Pastry, they replicates the files on a set of k nodes that have the `nodeld` nearby numerically to the key. An example of these applications is PAST [9] that replicates its files to assure a high availability in spite of failures in the nodes. Thanks to these properties of Pastry, the node that reaches a message in the next step is the most nearby one numerically in `nodeld`, and the most nearby one in terms of the proximity metric. This is very useful in applications as PAST, since to recover a file that be in a next node in metric, minimizes the latency and network load in the client.

2.3. FreePastry

2.3.1. Introduction

Two implementations of Pastry are currently available for download: FreePastry from Rice University and SimPastry/VisPastry from Microsoft Research. The initial releases of both implementations support a semantically similar API. A joint, language independent API for Pastry is currently being defined and will be supported by future releases of both implementations. FreePastry has chosen because is implemented in java while Simpastry/Vispastry is based on C.

FreePastry [8] is an open-source implementation of Pastry intended for deployment in the Internet. The initial release of FreePastry is intended primarily as a tool that allows interested parties to evaluate Pastry, to perform further research and development in P2P substrates, and as a platform for the development of applications. Plans for later releases are to provide a fully secure implementation that is suitable for a full-scale deployment in the Internet.

2.3.2. Node creation

To create a node in FreePastry the method `newNode` (NodeHandle Bootstrap) of the class `PastryNodeFactory` is used. To this method is necessary to pass him the object `NodeHandle` of the node that will do of bootstrap. An object `NodeHandle` serves to handle to a node remotely, for example, to obtain its `Nodeld`.

The method `newNode` returns directly an object `PastryNode` that represents the node that has been created. All the process of join is carried out of automatic form inside the own program FreePastry.

2.3.3. Sending a message

There are two forms to send a Pastry message Pastry: the first one is to route the message by the Pastry network according to a key given; the second is that be sent the message directly to destiny. This last option is used when the node

destiny is already known and the message wants to be sent directly toward him, without they have that to use the state tables neither the routing algorithm of Pastry (see Fig. 2.4).



Fig 2.4 Sending directly a JoinRequest message from A to B.

In FreePastry to send a message in this way is used the method `recieveMessage (Message msg)` of the class `NodeHandle`, in the figure 2.2 an example of sending is shown. To this method not one must pass him any type of key, only one must pass him the message that wants to be sent. An object of the class `NodeHandle` always goes connected with a Pastry node, and serves to the other nodes to manipulate diverse aspects, for example: to obtain the `nodeId` of the node, to do that receive a message, to know if the node is alive...

On the other hand, if the destiny is not known, or the node wants to route a message throught the Pastry network, the message is must to put in a Pastry message of type `RouteMessage`, and to send it by means of the method `routeMsg (ID key, Message msg, Credentials cred, SendOptions opt)`. The destiny of the message will be the node that has the `nodeId` more numerically closest to the key. In the figure 2.5 the routing of a `JoinRequest` message is shown, the node destiny will be B due to that the key of the message is the identifier of B.

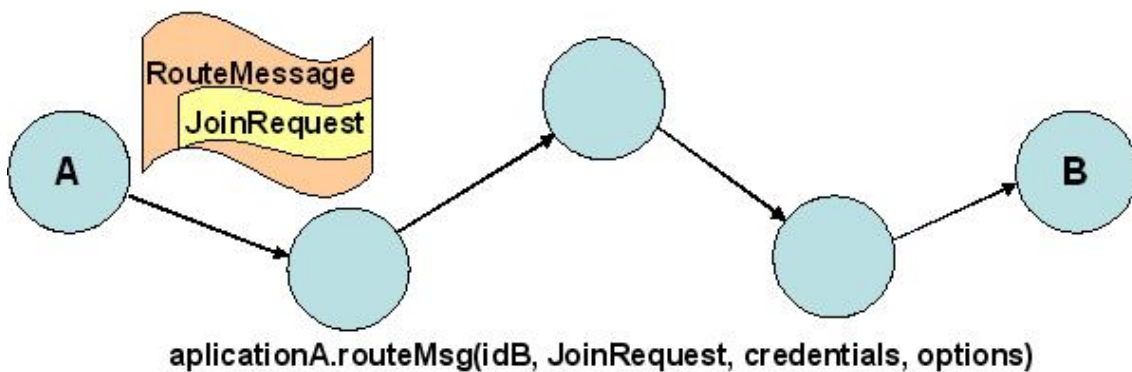


Fig 2.5Route Message

2.3.4. Implementation of an application on FreePastry

To create an application on FreePastry it is necessary:

- Create a class that inherit of the PastryAppl class.
- Besides an Address class must be created, this class is specific for the new application.
- Each Pastry node that wants to use this new application should be incorporated in a new object of this application.

By means of this new class the nodeld of the node can be agreed, also its Leafset, its RouteTable, etc... As well as also to send and to receive messages.

2.4. Past

PAST [9], a large-scale peer-to-peer persistent storage utility. Past is an application that runs on the nodes of the Pastry network. PAST is based on a self-organizing, Internetbased overlay network of storage nodes that cooperatively route file queries, store multiple replicas of files, and cache additional copies of popular files. It implements a DHT (Distributed Hash Table) allows the storage with replication of objects among the nodes that form the Pastry network.

The DHT are an implementation of the Hash Tables (HT) in distributed and not centralized environments. Its function is storing information, of any type, carrying out an association Key-Data. To agree to the information, the data, only we need to know the Key that has associate.

2.4.1. Characteristics

The PAST system is composed of nodes connected to the Internet, where each node is capable of initiating and routing client requests to insert or retrieve files. Optionally, nodes may also contribute storage to the system. The PAST nodes form a self-organizing overlay network. Inserted files are replicated across multiple nodes for availability. With high probability, the set of nodes over which a file is replicated is diverse in terms of geographic location, ownership, administration, network connectivity, rule of law, etc.

Past permits to a user to store an object in the Pastry network of a distributed form. This object should inherit of the ContentHashPastContent class. So that all the objects stored with Past are ContentHashPastContent objects. In the PAST system, storage nodes and files are each assigned uniformly distributed identifiers (160bits in FreePastry), and replicas of a file are stored at nodes whose identifier matches most closely the file's identifier.

Past stores any object permitting you replicate in diverse nodes. The number of desired replicas (k) is determined in the Past implementation:

PastImpl(Node node, StorageManager manager, int k, java.lang.String instance)

This method returns the Past object for a node, with this Past object the ContentHashPastContent objects can be inserted and recovered.

The most significant drawback of Past is the fact of be able modify/eliminate no object once inserted.

2.4.2. Insertion of a new element in Past

First the Past object that wants to be stored should be created, subsequently to create an identifier for that object and finally the node must to call the insert method through the Past instance:

insert(PastContent obj, Continuation command)

During an insert operation, PAST stores the file on the k PAST nodes whose nodelds are numerically closest to the 160 bits of the file's fileld. This invariant is maintained over the lifetime of a file, despite the arrival, failure and recovery of PAST nodes. For the reasons outlined above, with high probability, the k replicas are stored on a diverse set of PAST nodes.

This guarantees that any node can recover the inserted element if knows the identifier associated to the object and besides that at least one of the nodes that contain a replica be active in Pastry.

2.4.3. Recover a Past object

To recover a Past object is necessary to know previously the associated 160 bits identifier to that object. If the identifier is known only remains to call to the function of the Past class:

lookup(Id id, Continuation command)

This method retrieves a copy of the file identified by fileld if it exists in PAST and if one of the k nodes that store the file is reachable via the Internet. The file is normally retrieved from a live node "near" the PAST node issuing the lookup (in terms of the proximity metric), among the nodes that store the file.

2.4.4. Replica Diversion

The responsibilities of the storage management are to (1) balance the remaining free storage space among nodes in the PAST network as the system-wide storage utilization is approaching 100%; and, (2) to maintain the invariant that copies of each file are maintained by the k nodes with nodelds closest to the fileld. Goals (1) and (2) appear to be conflicting, since requiring

that a file is stored on k nodes closest to its fileid leaves no room for any explicit load balancing. PAST resolves this conflict in two ways.

First, PAST allows a node that is not one of the k numerically closest nodes to the fileid to alternatively store the file, if it is in the leaf set of one of those k nodes. This process is called replica diversion and its purpose is to accommodate differences in the storage capacity and utilization of nodes within a leaf set. For example, if a node A cannot store the replica, this node must to pass the replica to another node B. So that the node A maintains a pointer to the node B to redirect all the Past requests that receive on that object that now has the node B. Replica diversion must be done with care, to ensure that the file availability is not degraded.

2.4.5. File Diversion

The other solution is called File Diversion, it is performed when a node's entire leaf set is reaching capacity. Its purpose is to achieve more global load balancing across large portions of the nodeid space. A file is diverted to a different part of the nodeid space by choosing a different salt in the generation of its fileid.

File diversion consists of recalculate the identifier of Past object, so that other number of replicas are chosen more nearby nodes to identifier of object Past. If after 3 File Diversion not yet the object has been able to be stored Past is sent a message to emitter indicating that has been impossible to store the Past object.

CHAPTER 3. PKI-P2P application

3.1. Introduction

The purpose of this project is to implement an application that have the functionalities of a PKI and that function on a P2P network, therefore the name is PKI-P2P. The motive to utilize a P2P network is for the advantages that can contribute due to the decentralization of the information. These advantages are a great failure resistance, load distribution, storage with redundancy and it can be self-organized. These characteristics help favorably to the security and operation of a PKI.

The main objective of PKI-P2P is to permit to a user to verify that a public key is of whom says to be, permitting thus sure transactions of information between two users. In this chapter all the elements that form the application PKI-P2P as is explained in detail, well as also its procedures and functionalities.

3.2. Structure and design of the application

The implemented PKI model is based on the Thomas Wölf document [2]. In this document Thomas Wölf describes the protocol and procedures of a PKI on a Chord P2P network P2P, besides the characteristics of the PKI are based on the model of Maurer [2]. According to the document, this model is used due to its strong theoretical bases and its ability to shape the relations of confidence. An important aspect on the Maurer model is that does not speak of revoke/expiration of the certificates, so that the application PKI-P2P does not have la certificates revoked list neither controls the expiration of these.

PKI-P2P is based on the network Pastry instead of Chord owed in greater part to a greater knowledge of Pastry and its software already implemented in java (FreePastry). All the application this programmed in java and contains three representative main classes of each one of the functional layers that has the application. These classes are PKIApp, P2PApp, PKICipher (see Fig. 3.1).

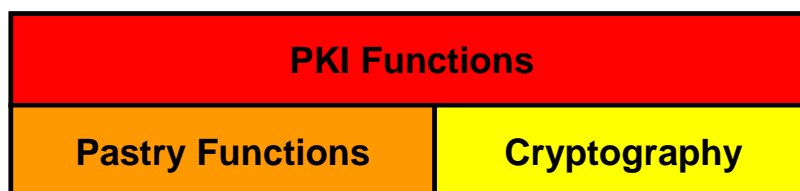


Fig 3.1 Layers

PKICipher contains all the methods and cryptographic techniques used by PKI-P2P. P2PApp contains all the methods and functionalities that allow the communication between nodes through the Pastry network. And finally PKIApp contains all the own methods of a PKI.

3.2.1. Cryptographic techniques used

PKI-P2P uses all types of cryptographic techniques to carry out its assignment. Mainly it is based on the asymmetric key algorithm so that each node of the network should have a key pair: a public key and a private key. These keys are generated at start of the application using the asymmetric algorithm RSA and they are kept in a directory called keystore. To generate the keys makes use of the KeyPairGeneration class and its generateKeyPair() method, to this class only one must indicate it that algorithm wants to be used and the length of the keys in bits.

The public key is stored in X.509 format and with *.publica extension. The X.509 codification is carried out by means of the class X509EncodedKeySpec. This class represents the ASN.1 encoding of a public key, encoded according to the ASN.1 type SubjectPublicKeyInfo. The SubjectPublicKeyInfo syntax is defined in the X.509 standard as follows:

```
SubjectPublicKeyInfo ::= SEQUENCE {  
    algorithm AlgorithmIdentifier,  
    subjectPublicKey BIT STRING }
```

Fig 3.2 SubjectPublicKeyInfo syntax

On the other hand, the private key is codified in PKS8 format and is stored with *.privada extension. The PKS8 codification is necessary because all the security of PKI-P2P falls in the private key of each node, nobody except the local node should know this key. The codification is carried out using the PKCS8EncodedKeySpec class. This class represents the ASN.1 encoding of a private key, encoded according to the ASN.1 type PrivateKeyInfo. The PrivateKeyInfo syntax is defined in the PKCS#8 standard as follows:

```
PrivateKeyInfo ::= SEQUENCE {  
    version Version,  
    privateKeyAlgorithm PrivateKeyAlgorithmIdentifier,  
    privateKey PrivateKey,  
    attributes [0] IMPLICIT Attributes OPTIONAL }  
  
Version ::= INTEGER  
  
PrivateKeyAlgorithmIdentifier ::= AlgorithmIdentifier  
  
PrivateKey ::= OCTET STRING  
  
Attributes ::= SET OF Attribute
```

Fig 3.3 PrivateKeyInfo syntax

To cipher messages or text the symmetric key algorithm DES is used because the computational cost of encrypt with RSA is more raised than with a symmetric key algorithm as DES. To share the secret key makes use of a hybrid system DES-RSA, where the secret key is encrypted with the public key of the receiver so that only the receiver can decipher it. The secret key is generated by means of the class SecretKeyFactory to which only should be passed the symmetric key algorithm used. To cipher and to decipher the Cipher class is used, this class is generic because from there the ciphers and the decipherers for RSA and DES are obtained. The ciphers encrypt an array of bytes that could be a text, and they return another array of completely intelligible bytes.

Also it makes use of Hash functions, in PKI-P2P the Hash function used is MD5. By means of the MessageDigest class can be done the checksum of an array of bytes, the result is a block of bytes, and it does not matter the length of the input bytes array, because MD5 will generate always a block of bytes with the same length.

Besides using multiple algorithms, PKI-P2P permits to sign digitally an array of bytes (for example a text, or a code ...). The signature is carried out with the RSA private key of the user, and the result is an array of intelligible bytes. The class that permits to sign digitally is Signature by means of its sign () method. With this method a structure of data is created to transport the signature, this structure is implemented in the SignedData class and contains: the digital signature, the public key of the issuer, the nodeld of the issuer, the nodeld of node of the subject. In each certificate and recommendation is included an object SignedData doing the function of digital signature.

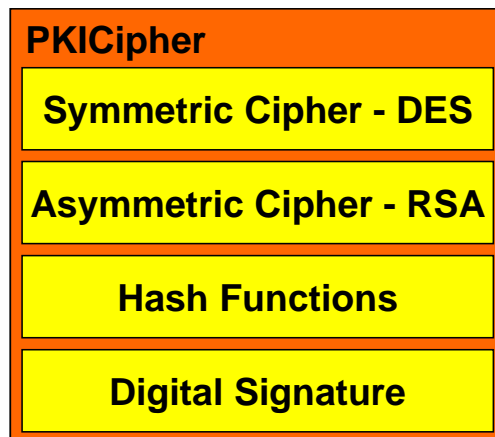


Fig 3.4 PKICipher class

All the methods that permit to carry out all the cryptographic functions previously mentioned are implemented inside the PKICipher class (see Fig. 3.4); this class is one of the three layers that constitute the PKI-P2P program.

3.3. PKI-P2P elements

The PKI-P2P users have the objective to maintain a sure communication, for it, the most critical point is to ascertain if the public key that obtain of the recipient, really is of the recipient. In this public key authentication process to take part various elements that will mention in this section and that are necessary to carry out the key authentication.

As it has been able to be seen in the previous section, in PKI-P2P has several cryptographic techniques to maintain insurances the transactions among the users. Besides these techniques there are other elements that have been implemented and are necessary, for example the messages that exchange the nodes or the objects that represent the confidence that has a user before the others, or the protocol communications that follow the nodes, the storage of the data,... In the next sections all these elements that to take parts in PKI-P2P are explained.

3.3.1. User identifier

Each user of PKI-P2P has a unique identifier, this identifier is the same one utilized for the Pastry nodes. This identifier is of 160 bits and is implemented in the Nodeld class of the FreePastry package. Using the same identifiers of Pastry we assure us that for each Pastry node there can be a PKI-P2P user.

3.3.2. Public messages

Each use of PKI-P2P can generate some objects that are public; this means that these objects are accessible by other users of the network. According to the Thomas Wölf article these objects are called public messages, which represent the only information communicated between nodes of PKI-P2P, but in PKI-P2P these public messages are inserted directly in Past to be able to share them with the remainder of the nodes. There are two types of public messages:

- Certificate: This message represents a digitally signed certificate, issued by node X. The certificate attests the binding of node Y (subject of the certificate) to public key PY. The digital signature of the message is valid, if node X is the authentic owner of public key PX, i.e. the digital signature can be validated via public key PX.

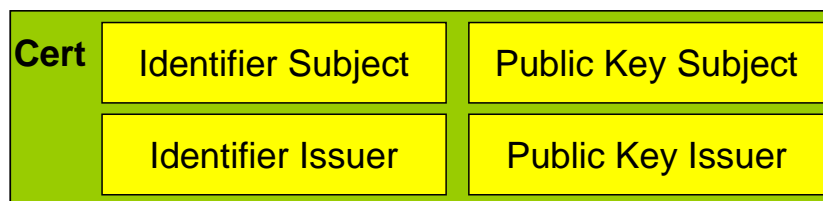


Fig 3.5 Cert

- Recomendaciones: This message type is used to transport a recommendation of level i for node Y (subject). It is issued and digitally signed by node X. The digital signature of the message is valid, iff node X is the authentic owner of public key PX, i.e. the digital signature can be validated via public key PX. The purpose of this level i is to indicate the level of confidence that a node X has in node Y.

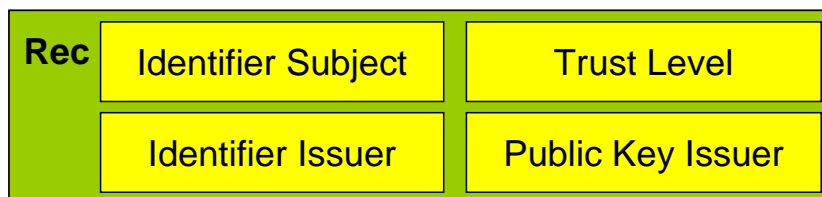


Fig 3.6 Rec

These public messages are not transmitted in clear, when a node wants to transmit a certificate or a recommendation ciphers the message by means of a symmetrical key algorithm. Besides each public message has associated a key (not the cipher key) that is used to generate the index of public message. This index is unique, and is used when a node wants to seek a certificate or a recommendation of a certain user.

Once it a public message is encrypted and generated its index all message is included inside another structure of data. This new object is called "Message Token" (see Fig. 3.7), and is the real message that exchanges the nodes of the PKI-P2P.

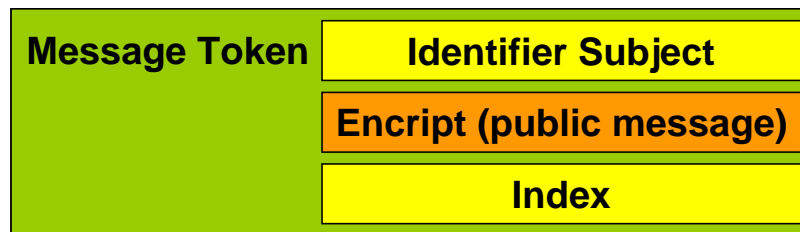


Fig 3.7 Message Token

The public messages of the PKI-P2P are signed for nodes, there is no type of certification element of confidence type CA, in this model of PKI all the nodes are equals and they can sign so many certificates and recommendations as they desire.

3.3.3. Private Statements

Private statements are used to describe a node's belief in other nodes of PKI-P2P. They are not accessible for any other node except the local one, which produces these statements. There are two types of private statements in PKI-P2P:

- Authenticity (Aut): This statement type denotes the local node's conviction that node X is the authentic owner of public key PX. Aut contains a nodeld of a node X and its public key.
- Trust: This message type denotes the local node's trust in a certain node X with level i. It is used to represent the node's degree of belief in node X to act in accordance with its security requirements.

3.3.4. Private View and Public View

So much the public messages, as the private objects should be stored in some place. Each node has two spaces where to keep these elements, the private view and the public view. In the private view each node keeps the objects authenticity and trust and in the public view the certificates and recommendations are kept.

The private view is easy to implement since can be perfectly a folder by name "private view" where the node keep the private statements because the node hasn't to share with nobody. But in the case of the public view is something more complicated to implement because the public messages should be visible and accessible for all the nodes. The way to do that all the public messages is

accessible for all the nodes of PKI-P2P is utilizing Past (the DHT implemented of Pastry).

With Past a node can insert a public message and to share it with the remainder of the nodes of the network. The DHT already takes charge of distributing fairly the public messages among the nodes that compose the P2P network. Past also creates automatically a folder in the local node where will keep the assigned public messages.

In summary, each node will have two folders, one the private view that consists of a folder that stores the private statements, and the other the public view that is a folder that is created for Past automatically where the public messages are stored (see Fig. 3.8).

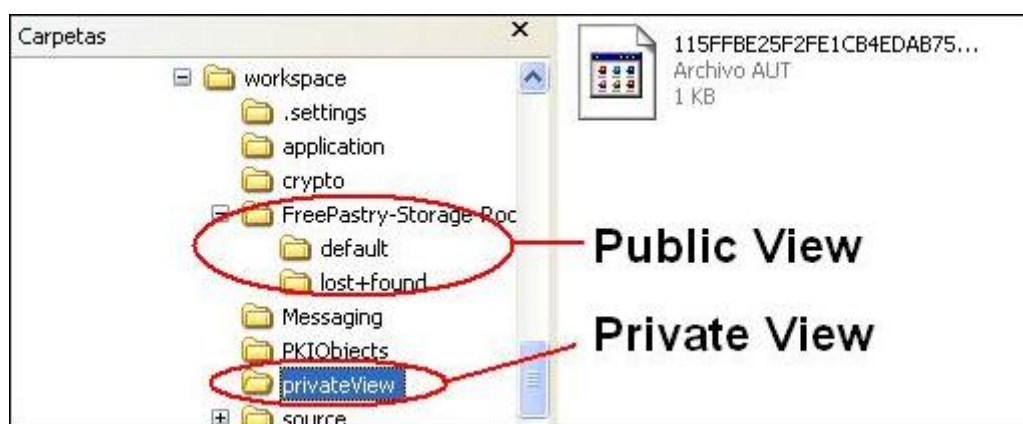


Fig 3.8 Private and Public View

3.3.5. PAST in PKI-P2P

The public messages are inserted in an object of the MessageToken class, these objects can be emitted for each user of the network and they should be at the disposal of any user, are public. For it the storage of this type of objects is distributed among all the PKI-P2P nodes by means of the utilization of the DHT Past. A Past object is an instance of the PastContent class, so that any object that be wanted to insert in Past should inherit of that class. In PKI-P2P there are three types of objects that are distributed to the other nodes by means of Past: the Message Tokens, the public keys of each node, and the digital signatures that does each user on a certificate. Each one of them it is represented in the PKIPastContent, KeyPastContent and PruebaPastContent classes respectively and the three inherit of PastContent.

In PKI-P2P three functions of Past are utilized that basically are: insert (), lookup () and exists ().

- Insert (): this method is used when, for example, a MessageToken wants to be inserted in Past.

- Lookup (): when wants to recover for example a MessageToken of Past.
- Exists (): permits to know if a specific object already exists in Past from its identifier, because in Past only there can be an object by identifier.

3.3.5.1. Insertion of a public message

To insert a public message in Past, first of all and by security reasons, the node should encrypt the certificate or recommendation by means of an symmetric key algorithm and to include it in a Message Token next to an message index. The indices of message are unique for each public message and they are generated of two different ways as is a matter of a certificate or recommendation:

- For a certificate, a Hash function must apply to the identifier of the node (nodeld) more the public key of the node. The secret key for cipher the public messages is generated from the result of Hash function. And finally it applies a Hash function to the secret key generating thus the index of the public message.

```
digest = Hash( Nodeld_Bob + PublicKey_Bob )
SecretKey = genSecretKey ( digest )
Index_k = Hash (SecretKey )
```

Fig 3.9 SecretKey and Index k generation for a certificate

- For a recommendation one must continue the same steps that for a certificate, but the first Hash function only applies to the nodeld because in a recommendation does not have public key.

```
digest = Hash( Nodeld_Bob )
SecretKey = genSecretKey ( digest )
Index_k = Hash (SecretKey )
```

Fig 3.10 SecretjKey and Index K generation for a recommendation

The node that want to insert a Message Token in Past should use the place_msg_token(MessageToken mtoken) function of the P2PPKI class (or to lower level the insert () method of the PastImpl class). The Message Token objects inherit of the PastContent objects so that a Message Token is apt for to be distributed in Past. The index of public message generated will be utilized to generate the identifier that associates to Past object inserted.

3.3.5.2. Recovery of a public message

Para recuperar un mensaje público se debe generar el identificador de objeto Past ligado a ese mensaje público. Para ello se deben de seguir los pasos descritos en el apartado anterior, por ejemplo para recuperar un certificado:

To recover a public message the identifier of Past object should be generated, this identifier is linked with the public message. To that purpose the steps described in the previous section should be continued, for example to recover a certificate:

- Checksum = Hash (nodeld + public key of this node).
- Secret Key = encrypt(checksum) This secret key allows to decrypt the Message Token once is recovered.
- Index of Message Token = Hash(secret key)
- With this index the identifier necessary to recover the Message Token is generated.

The only requirement to obtain a certificate is to know the public key beforehand. If the user doesn't has the public key, this can be obtained of Past before recover a Message Token.

3.3.5.3. Digital Signatures

Due to that in Past an object must have only an identifier, is not possible to have various certificates with different digital signatures of a determined node, a node can have only a digital certificate. It is neither possible that a certificate contain all the digital signatures, because in Past the objects cannot be modified. So that the digital signatures also are inserted in Past, instead of including the digital signatures inside a certificate.

When a user A wants to sign a certificate of a user B creates a SignedData object (representative of a digital signature). This object is inserted in Past. So that when a user wants to verify the digital signatures of a certificate will have to obtain of Past the certificate and each one of the digital signatures.

3.3.5.4. Communication between nodes

Besides being communicated for Past with the public messages, the nodes need a communication more direct node to node without any DHT by middle. For example just before sending an encrypted message with DES to a node, this node should know the secret key, therefore this key must be sent it through the Pastry network. Or another case is once has been verified that the public key of a node is authentic; all the next communication is done through conventional messages through Pastry. Exactly for these two cases two messages has been defined:

- **SimetricKey**: is utilized to transport a DES secret key encrypted with the public key of the recipient so that only the recipient can decipher the content of the message and to obtain the secret key.
- **TextMessage**: this message contains text encrypted with DES. These messages are used to exchange text between two users of sure form, so that all the text that is sent cannot be read by third people.

3.4. Trust Model

In this section the functionalities utilized to authenticate a public key are detailed.

3.4.1. Insertion of the public key

Once it starts the application, if is the first time that a node agrees to the Pastry network, the RSA key pair are created for that node and the public key is inserted in Past to share it with the remainder of users of the network. For the insertion in Past the public key is put in a KeyPastContent object, and the identifier of this Past object is generated from the identifier of Pastry node (nodeId).

3.4.2. Self-signed certificates and digital signatures

Besides creating the RSA key pair a self-signed certificate is created with the public key of the node. A self-signed certificate is when a user signs itself digitally its own certificate. The utility that have these self-signed certificates is that of guaranteeing a "Trust Anchor" in a chain of certificates. As already it has been explained previously, in PKI-P2P the nodes are the responsible for creating chains of certificates, any node can sign digitally a certificate. So that when a node A receives a certificate of B that contains a firm of C, the node A should verify that really the firm is of C. Therefore the node A should obtain the certificate of C that this signed by the node D, and again will have to verify the firm of C obtaining the certificate of D that this signed by E, and so on what it creates a chain of certificates. The final element of this chain is called "Trust Anchor".

A self-signed certificate is a "Trust Anchor" because the digital signature is of the own node to that the certificate refers, the chain cannot be continued. In the case to obtain a self-signed certificate, the decision to trust or not in that digital signature falls in the user of the application. With these self-signed certificates a "Trust Anchor" is assured in these chains of certificates (see Fig. 3.11).

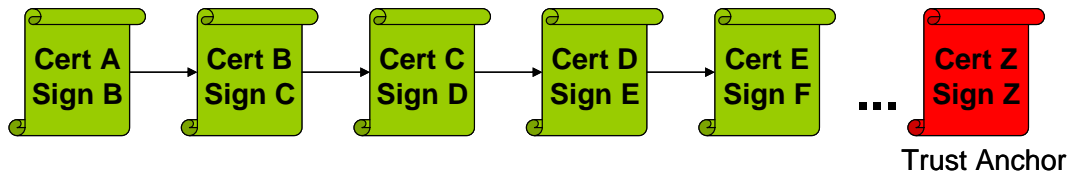


Fig 3.11 Chain of certificates

Other "Trust Anchor" elements are the Aut private objects of PKI-P2P, these objects indicate full confidence of a user toward another user with a determined key public. So that if a certificate of the node B is obtained with a certain public key and looking at you Private View is found an Aut of B with the same public key, does not do lack to verify the digital signature of the certificate because already is trusted fully in that public key.

3.4.3. Authentication process

At this moment a user can establish a sure communication with another user of the Pastry network that have the software PKI-P2P. For example a user Alice wants to send a "Hello!" message to a user Bob of sure form. First Alice should obtain the public key of Bob (see Fig. 3.12) and to verify if really belongs to said user. To obtain the public key, Alice does a petition to the DHT Past of the Past object with identifier the Bob Nodeld. Past will return to Alice a KeyPastContent object that contains the key of Bob. Now Alice already has the public key of Bob, only remains to verify if the key is really of Bob.

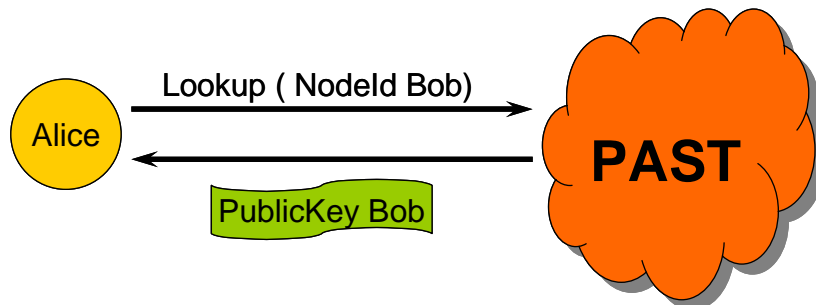


Fig 3.12 Process to obtain the public key of Bob

To verify the key the `is_authentic` method of the `PKIApp` class is called. This method verifies if a public key is authentic or not. If the public key is authentic this key is saved in the keystore. This method follows the next steps:

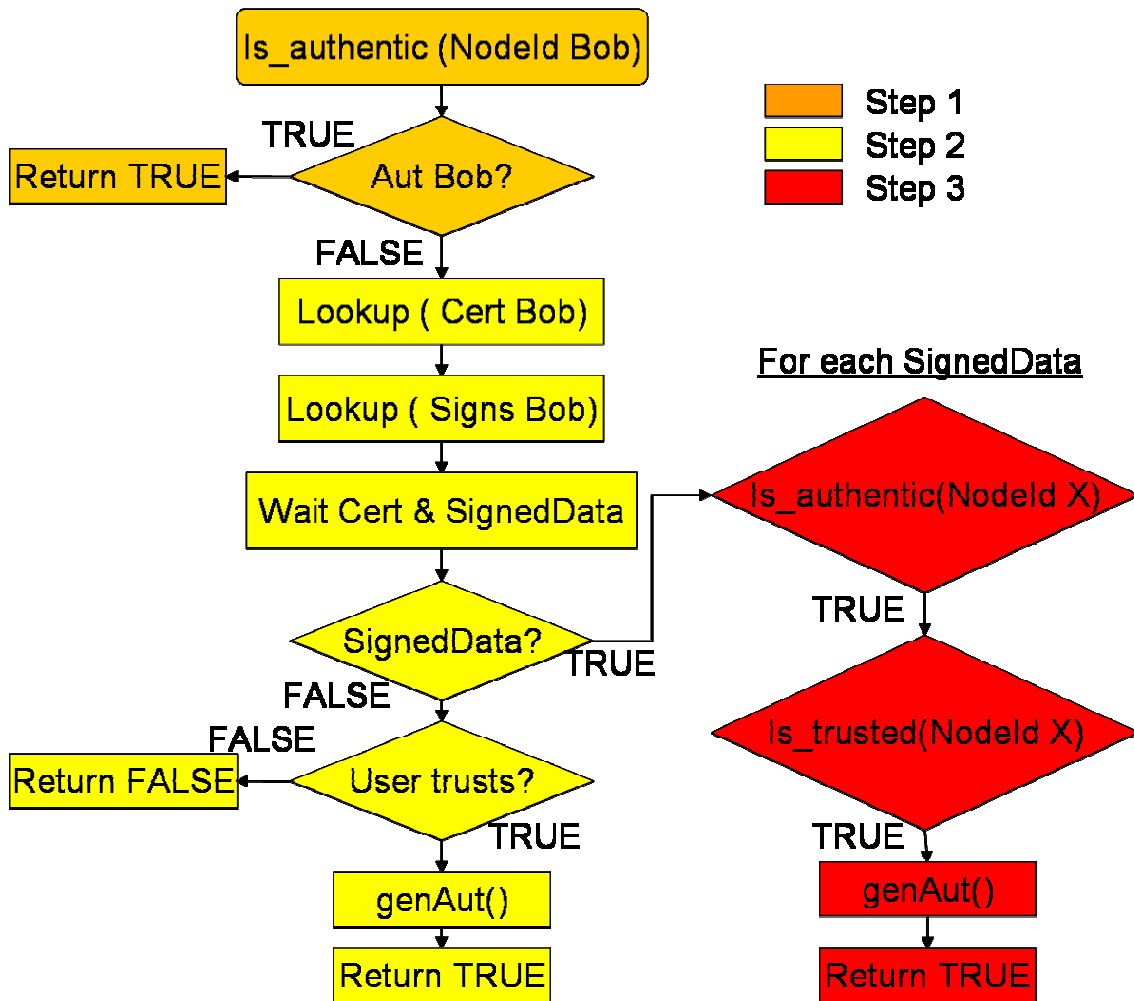


Fig 3.13 Authentication process

Step 1: Looking at Private View

First Alice look at if has already an Aut of Bob, because an Aut indicates that Alice trusts fully in that public key. To know if an Aut exists the `is_authenticity_contained()` method is called and returns TRUE or FALSE according to if the Aut has been found or not. If an Aut of Bob is found, the public key is kept in the Keystore. If an Aut doesn't is found, Alice must passed to step 2.

Step 2: Recovery of a certificate of Bob and all digital signatures

To obtain the certificate a petition to Past is done, the necessary identifier is created from the `nodeld` of Bob and its public key. At the same time that the petition of the certificate is done the petition of its digital firms is done. The certificate that is received is a self-certificate; this means that has been signed by Bob. Subsequently the digital signatures are obtained, that are formed as a SignedData objects.

Now may pass two things: Alice receives the digital signatures or there are not digital signatures in Past. If Alice have not been received a digital signature,

only remains to trust in the self-certificate of Bob, this decision remains in the user of the application and what causes is to carry out the question: "¿It trusts in the self-certificate of Bob? yes/no". If Alice decides to trust in the self-certificate will be kept the public key of Bob in the keystore and an Aut of Bob will be generated, otherwise the public key is considered not valid/authentic.

Step 3: Verify of the digital signatures

If Alice has received some digital signatures, a signature is chosen. Alice verifies that the issuer of the signature is who says to be, this causes calling of the `is_authentic()` method, so that each element of the chain of certificates is authenticated. Until a "Trust Anchor" is not found the authentication process does not finalize for each element of the chain of certificates. If the issuer of certificate of Bob can not be authenticated, another signature is chosen and the process is repeated. If do not remain firms, the public key of Bob is considered not valid/authentic.

If a user X is found that has digitally signed and is authentic, subsequently Alice should look at if that user X is of confidence in an equal or greater level that 1. For it the `is_trusted()` method is called. This method begins calling the `is_trusted_contained()` method to look at if Alice has a Trust object of the user X in the Private View. A Trust object contains the user identifier and the level of confidence that has Alice in that user.

If a Trust object of user X is found, Alice should to verify that the level of that Trust is greater or equal as 1. So that if complies that:

- The user X that signs the public key of Bob is really that user.
- If Alice trusts in X in a greater level or equal to 1.

Alice will give for authenticates the public key of Bob and will keep it in its keystore, besides will generate an Aut of Bob with that public key.

In the case that Alice's Private View hasn't a Trust object of user X, a recommendation of X should be obtained of Past and the digital signatures associates to that recommendation. The recommendation that is obtained is a self-signed recommendation by X. If is impossible to obtain a digital signatures for the recommendation of X, the application asks to Alice if trusts in the self-signed recommendation, the decision to generate a Trust fall in the user of the application.

If signs arrive Alice should looks a field that indicates the trust level that has the issuer in the node X. Alice must seek a user that trust in X in a greater level or equal to 2 and that be possible to verify its authenticity by means of the `is_authentic()` method. If a user that comply these conditions is found an Trust object of user X will be generated. If Alice manages to generate a Trust of X will be able to verify that Alice trusts in the user X in a greater level or equal to 1 and will be able to generate an Aut of Bob.

Continuing these three steps Alice can determine if the public key of Bob received is of confidence or not.

3.4.4. Private messages shipment

Once a user has trusted in a public key will make use of this key to send for example private messages. But the private messages will not be encrypted directly with the RSA algorithm and with the public key but makes use of a hybrid DES-RSA algorithm. In this hybrid algorithm a DES secret key is generated and is shared with the other user making use of the RSA algorithm, signing the secret key with the public key of the recipient. The recipient will make use of its private key for decrypt the secret key, and with this key the nodes can begin the shipment of private messages.

When a user wants to send a private message to another user should encrypt that message with DES using the secret key shared previously. So that the recipient will be able decrypt with its copy of the secret key, only they two are able decrypt the sent messages because only they two have a copy of the secret key. The shipment of the secret key among user is carried out by means of a Pastry message described in the `SymmetricKey` class. And the shipment of encrypted messages with DES is carried out by means of Pastry messages described in the `TextMessage` class.

CHAPTER 4. PlanetLab

4.1. Introduction

PlanetLab [10] is a global network to give support to new services that are developing at present. PlanetLab was launched in 2003 and since then more than 1000 researchers at top academic institutions and industrial research labs have used it, mainly to develop new technologies and services for distributed storage, P2P networks, Distributed Hash Tables, network mapping and query processing.

Nowadays PlanetLab is formed by 780 nodes distributed in 382 points of the planet (see Fig. 4.1)., where the majority are public institutions such as Princeton University, Cambridge University or UPC. Besides also they form PlanetLab some investigation centers like Intel Research or HP Labs.

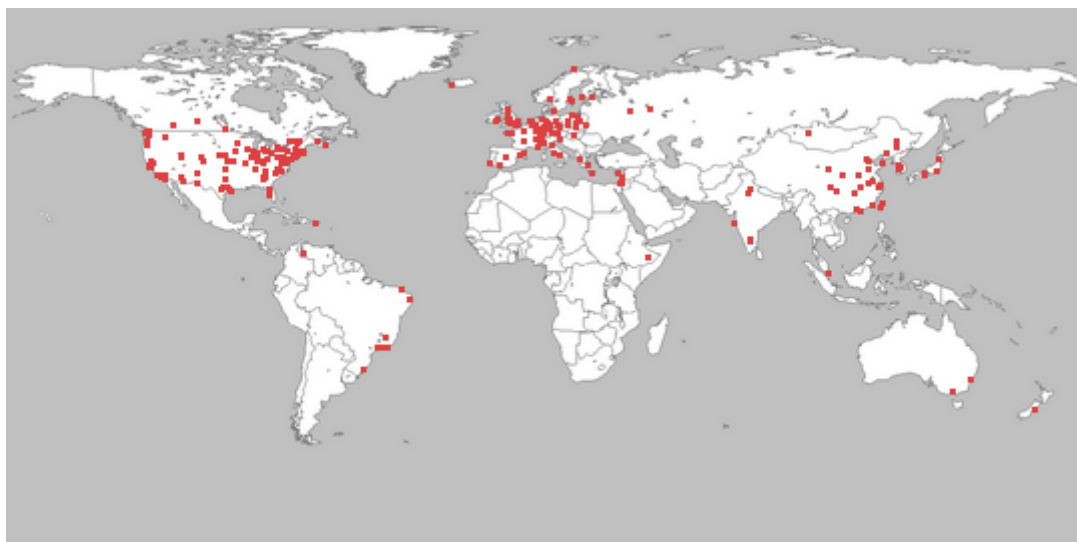


Fig 4.1 Institutions and industrial reserach labs

All these centers and institutions form the PlanetLab Consortium. The PlanetLab Consortium is a collection of academic, industrial, and government institutions cooperating to support and enhance the PlanetLab overlay network. It is responsible for overseeing the long-term growth of PlanetLab's hardware infrastructure; designing and evolving its software architecture; providing day-to-day operational support; and defining policies that govern appropriate use. The PlanetLab Consortium is managed by Princeton University, the University of California at Berkeley, and the University of Washington. Princeton currently hosts the Consortium. Larry Peterson currently serves as the Consortium's Director. Institutions join the Consortium by signing a membership agreement and connecting two or more nodes to the PlanetLab infrastructure. Individuals that want to use PlanetLab should arrange to do so through their home institution.

Each node of PlanetLab has installed common software that works on Linux. This software takes charge of monitoring the state of the node, its activity, the resources, etc.

All in all, the main idea is to be able to reproduce Internet to scale to carry out a tests that would have an a lot more high price if were done in the real Internet.

4.2. Membership

The consortium includes five membership levels:

- Charter (\$300k annual dues)
 - o Permanent seat on Steering Committee.
 - o Unlimited number of slices.
 - o Access to PlanetLabe events, research papers, and working groups.
- Full (\$75k annual dues)
 - o Rotating seat on Steering Committee.
 - o 10 slices.
 - o Access to PlanetLab events, research papers, and working groups.
- Associate (\$25k annual dues)
 - o 2 slices.
 - o Access to PlanetLab events, research papers, and working groups.
- Sponsor (\$10k annual dues)
 - o Access to PlanetLab events and research papers.
- Academic (no annual dues)
 - o Seat on Steering Committee by invitation.
 - o 10 slices.
 - o Access to PlanetLab events, research papers, and working groups.

4.3. Elements

Subsequently the elements that to take part in the PlanetLab network are detailed:

- **Site:** A site is a physical location where PlanetLab nodes are located (e.g. Princeton University or HP Labs). Abbreviated versions of site names prefix all slice names.
- **Node:** A node is a dedicated server that runs components of PlanetLab services.
- **Slice:** A slice is a set of allocated resources distributed across PlanetLab. To most users, a slice means UNIX shell access to a number of PlanetLab nodes. PIs are responsible for creating slices and assigning them to their users. After being assigned to a slice, a user may then assign nodes to it. After nodes have been assigned to a slice, virtual

- servers for that slice are created on each of the assigned nodes. Slices have a finite lifetime and must be periodically renewed to remain valid.
- Sliver: A set of allocated resources on a single PlanetLab node.
 - **Virtual Server (Vserver):** Slivers are currently implemented as Linux-Vservers, which implements both namespace and performance isolation among slivers on a single machine.
 - **Principal Investigator (PI):** The PIs at each site are responsible for managing slices and users at each site. PIs are legally responsible for the behavior of the slices that they create. Most sites have only one PI (typically a faculty member at an educational institution or a project manager at a commercial institution).
 - **Technical Contact (Tech Contact):** Each site is required to have at least one Technical Contact who is responsible for installation, maintenance, and monitoring of the site's nodes.
 - **User:** A user is anyone who develops and deploys applications on PlanetLab. PIs may also be users.

4.4. Configuration

4.4.1. Use account

First of all an account of user must be created (see Fig. 4.2) in the web: www.planet-lab.org. For can be registered and to be a user of PlanetLab should belong to some of the adhering institutions, because is the PI of the center the responsible for to give out the slice to each user.

The image shows a web form titled "Account Registration". Below the title, there is a note: "Your E-mail address must be able to receive e-mail and will be used as your PlanetLab username." and a warning: "Do not select the **Principal Investigator** or **Technical Contact** roles unless you have spoken with the current PI of your site, and you intend to assume either or both of these roles." The form contains several input fields: "First name: *" (text box), "Last name: *" (text box), "Title:" (text box), "E-mail: *" (text box), "Password: *" (text box), "Telephone:" (text box), and "Site: *" (dropdown menu with "Select a site" as the current selection).

Fig 4.2 Form for user account

Once the registration this fact, the petition of registration is sent directly to PI, and if the PI seems him well gives you a Slice. To enter to Slice and to negotiate it one must authenticate with a login and a password (defined in the user account registration). Once a user is authenticated appears in the right part of the web a menu of PlanetLab user (see Fig. 4.3):



Fig 4.3 PlanetLab user menu

Since this menu is able:

- Sites: is a search engine of Sites besides the listing of all the existing Sites in PlanetLab. Selecting a Site permits you to see information on that Site: an abbreviated name, the URL, the login base, the latitude, the longitude, maximum slice count, maximum sliver count, technical contact, PI contacts, the address where the Site is found, the nodes that form it, and the Slice that have been created in that Site.
- My Site: permits to see information of your Site of the same form that with other Sites. Besides the name of your Slice appears here, this is a link since which you can access to your slice.
- Nodes: Here all the nodes that form PlanetLab can be seen nodes that. If click on a node the information of that node can be seen like: the institution they belong, which is its IP, its state, if there are more nodes in that institution, the name of host, etc.
- My Site Nodes: from here the nodes that belong to Site of the user can be seen and the state in which they are found. For example if my Site is the UPC will appear the nodes of PlanetLab that belong to the UPC.
- Add Node: from here a new machine can be added to Site.
- Users: is a search engine of the PlanetLab users, only the mail address of that user must put so that show you information about that user.
- Me: here all the information of our user is found: personal data, information of the RSA public key, the Site that belongs, the role and the Slice that is assigned to user (see Fig. 4.4).

Xavi Barrera Quintanilla Account Information

First Name: Xavi
 Last Name: Barrera Quintanilla
 Title: Mr
 Email: xavibq@gmail.com
 Phone:
 URL:

[Update info](#)

Keys

Type	Key
ssh-rsa	AAAAB3NzaC1yc2EAAAABIwAAAIEAx8Ey364mfpzloCuUYJMX/ZLc
ssh	cfr9EUizaOM6o/WYijjFU27ICGmZR74+zKe7mEkvUQsVDYj50pOZ u/so4LqqtROY/Ef63gd+d7ok22IMKjYOWYyxU3ZrNxyZI9aq9sAgJkI xbarrera@broadband110.upc.es

[Manage Keys](#)

Sites

[Universitat Politecnica de Catalunya](#) (remove)

Roles

Role
user

Slices

[upc_epsc](#)

Fig 4.4 User account information

- Slices: from here all the Slices that exist in PlanetLab can be seen, to whom they belong and a brief description of each one.
- Sirius: this option serves to choose hourly stripes for our Slice where will be given him a priority in terms of CPU.
- Log out: serves for log out of the system.

4.4.2. Slice expiration

A very important aspect in PlanetLab is that the user Slice is not infinite, has a validity period. So that the user must renew the Slice periodically. This is thus because the nodes of PlanetLab is a resource that is shared for all the users. So that if there is a user that time ago that does not work with its Slice is better to erase it and to leave the free resources for other users.

4.4.3. Nodes management

To be able to work with the PlanetLab nodes first one must add them one to one to our Slice. For it the user must go to the Me option, to select our Slice, and to choose the Manage Nodes option. In the Manage Nodes option appears all the nodes that have been assigned to Slice, and the nodes can be added or erased (see Fig. 4.5).

Slice upc_epsc - Nodes

Select a site to add nodes from.
University of Bologna

planetlab2.cs.unibo.it
 planetlab1.cs.unibo.it
Add Nodes

Nodes currently associated with slice
Check boxes of nodes to remove:

planetlab1.cs.columbia.edu
 planetlab2.cs.columbia.edu
 planetlab3.cs.columbia.edu
 planetlab3.nbgisp.com
 planetlab1.nbgisp.com
 planetlab-01.bu.edu

Fig 4.5 Add nodes in PlanetLab

Not all the nodes are equals, so that some nodes can be more loaded than other. Besides one must keep in mind the state in which these nodes are found, each node always is in one of these states:

- **Boot:** This state cooresponds with nodes that have sucessfully installed, and can be chain booted to the runtime node kernel.
- **Install:** The install state cooresponds to a new node that has not yet been installed, but record of it does exist. When the boot manager starts, and the node is in this state, the user is prompted to continue with the installation. The intention here is to prevent a non-PlanetLab machine (like a user's desktop machine) from becoming inadvertantly wiped and installed with the PlanetLab node software.
- **ReInstall:** In this state, a node will reinstall the node software, erasing anything that might have been on the disk before.
- **Debug:** Regardless of whether or not a machine has been installed, this state sets up a node to be debugged by administrators.

So that is important that the nodes that be added to our slice has the boot state. The state of the nodes can not be known from My Site Nodes option, the user must go to the Nodes option. In that section a list of all the PlanetLab nodes and their states is found (see Fig. 4.6).

Nodes

Enter hostname or pattern:

Nodes	
Boot_state	Hostname
boot	plab1-c703.uibk.ac.at
boot	plab2-c703.uibk.ac.at
boot	plab1-itec.uni-klu.ac.at
boot	plab2-itec.uni-klu.ac.at
boot	michelangelo.ani.univie.ac.at
boot	supernova.ani.univie.ac.at
boot	planet-lab-1.csse.monash.edu.au
boot	planet-lab-2.csse.monash.edu.au
rins	planetlab1.it.uts.edu.au
rins	planetlab2.it.uts.edu.au
boot	plnode01.cs.mu.oz.au

Fig 4.6 All nodes in PlanetLab

This implies that the nodes must to be added manually one to one of manual, the great objection is that if a user desire to do a test with 400 nodes, the user must repeat the operation to add a node 400 times. Besides the user should be sure that the nodes are in the boot state.

4.4.4. Nodes access

The access to the PlanetLab nodes is carried out by means of SSH. Therefore it must to have a SSH client besides generating a RSA key pair to be authenticated in the nodes of the system. To create a RSA key pair the program `ssh-keygen` can be used:

```
ssh-keygen -t rsa -f ~/.ssh/<nombre_clave>
```

The `ssh-keygen` program will ask for a passphrase, is recommendable by themes of security that has 10-30 characters. The `ssh-keygen` will generate a key pair: a public key with extension *.pub and a private key. The RSA public key must be distributed to the PlanetLab nodes of our slice, for it the user must upload the key to the PlanetLab web and the PlanetLab system already takes charge of distributing it. For upload the key the user must go to Me option and press on the Manage Keys option. Since there the key can be uploaded to the web or can erase an old key (see Fig. 4.7).



Fig 4.7 SSH public key

The distribution of the RSA public key RSA is not instantaneous, can delay some minutes. On the other hand, the RSA private key will be utilized to be authenticated in each SSH access. An example of SSH access:

```
ssh -l <nombre_slice> -i <clave_privada> <IP o nombre_Host>
```

Once done this command would be being agreed remotely to node. For each SSH access to the PlanetLab node the user must introduce the passphrase, if this wants to be avoided, a solution is to write the following commands sequence at start of each session:

```
eval 'ssh-agent'  
ssh-add
```

The objection of this access method is that if for example a test with 100 nodes wants to be done, and you want to agree to each one of these nodes, should do the SSH access one by one. This aspect complicates and does difficult the tests with a large quantity of nodes. The solution to this problem is use programs that permit SSH parallel accesses; an example of these programs is vxargs.

4.5. PlanetLab tests

One of the objections of PlanetLab is that the node to that agrees does not contain anything, for example the java virtual machine does not have it installed. So that the users should upload to the nodes all that is needed to execute their applications.

To copy the files and necessary programs to a PlanetLab node an option is to utilize the scp command. This command permits to copy files between two machines utilizing ssh for which offers the same security that ssh. In the case of this project one must copy to the PlanetLab nodes the java binary, the jar files and necessary configuration file, and the folder with the java classes. So that in this case the unique program that should be installed in the remote machines is the java.

An example of upload a file to a node would be:

```
scp freepastry.jar upc_epsc@131.232.34.56:freepastry.jar
```

Where upc_epsc is the name of the slice and 131.232.34.56 is the IP address of the remote node. When the user wants to copy a folder the option `-r` must put:

```
scp -r application upc_epsc@131.232.34.56:application
```

With this command the folder application and all its contained is copied to node with IP address 131.232.34.56.

The objection to do it by means of scp is that for each file or folder that be wanted to upload the user must do a command. So that to write a command for all the necessary files and later for each node is very thorny. The solution is the creating a script that does it all for each node, so that only one must write the script and then be to executed it only once. This script copies the files and necessary folders in a sequential and automatic form.

A script is a file written in shell language and with extension *.sh, that contains commands and structures. The scripts can be as simple as a succession of commands, or as complex as be wanted. With the text editor the scripts can be created, for example one very simple would be:

```
echo "hello world!"  
exit
```

For execute the script first the user must change the permission:

```
chmod 755 prueba.sh
```

Next execute it:

```
./prueba.sh
```

This test script the unique thing that causes is show the text "hello world!" in the linux terminal.

To copy files to the PlanetLab nodes and to install the java, the script used is not very complicated. Subsequently the script used can be seen:

```
for node in 138.232.66.194 200.19.159.35 192.41.135.218  
do  
scp jre-1_5_0_11.bin upc_epsc@$node:jre-1_5_0_11.bin $j2re  
scp installjava.sh upc_epsc@$node:installjava.sh  
scp FreePastry.jar upc_epsc@$node:FreePastry.jar  
scp log4j.jar upc_epsc@$node:log4j.jar  
scp bcprov-jdk.jar upc_epsc@$node:bcprov-jdk.jar  
scp xmlpull.jar upc_epsc@$node:xmlpull.jar  
scp xpp3-1.1.4.zip upc_epsc@$node:xpp3-1.1.4.zip
```

```

scp log4j.properties upc_epsc@$node:log4j.properties

scp -r application upc_epsc@$node:application
scp -r crypto upc_epsc@$node:crypto
scp -r distapplication upc_epsc@$node:disapplication
scp -r messaging upc_epsc@$node:messaging
scp -r PKIObjects upc_epsc@$node:PKIObjects
scp -r privateView upc_epsc@$node:privateView

ssh -x upc_epsc@$node "./installjava $j2re"
done

```

This script copies all the jar files, configuration files and necessary folders to execute the PKI-P2P application to the nodes 138.232.66.194, 200.19.159.35 and 192.41.135.218. Besides, the java is installed in the remote node.

To install the java makes use of the installjava.sh script that is executed directly in the remote machine by means of the ssh -x command. This script contains the following thing:

```

j2re=$1
echo Installing java.bin
echo "yes" > yes
sh $j2re < yes
rm yes
mv jre1.5.0_11/ java/

echo Setting Environments variables
echo "CLASSPATH=.:$HOME/java/lib
export CLASSPATH
JAVA_HOME=$HOME/java
export JAVA_HOME
JDK_HOME=$JAVA_HOME
export JDK_HOME
PATH=.:$JAVA_HOME
export PATH" >> .bashrc
source .bashrc

```

This script has been obtained of a PlanetLab tutorial and what causes is to install the java in a node besides configuring all the environment variables required.

All in all, if 20 nodes are needed to do a test, with these two scripts all necessary files are copied to those 20 nodes. Besides the java are installed, and leaves those nodes ready for execute the PKI-P2P application.

To execute the PKI-P2P application the best option is to execute it in a simultaneous way by means of vxargs. Because the scripts execute the commands in a sequential way and due to that the PKI-P2P application never finishes. When the PKI-P2P application is executed in first node, the script

would not pass never to be executed to a second node. Because in a script until a command doesn't finished the following command is not executed.

4.5.1. Vxargs: parallel ssh access

Vxargs [12] is a small application written in python that permits to execute ssh commands in parallel and simultaneous form in diverse nodes. Vxargs is executed from prompt and must pass him a text file with the IP addresses of the nodes that user wants to agree.

For example if you want to do a test with 100 nodes the vxargs program is very useful once those 100 nodes already have all that is needed to begin the test (that the nodes have the java installed, the necessary files, etc...). The vxargs in our case is utilized to execute the PKI-P2P application and also is utilized once finalized the test for stop the application in each node (although this has also done by means of script).

To execute vxargs only the user must put the following command:

```
python vxargs-0.3.3.py -a node.txt -o /home/result ssh -x upc_epsc@{  
"command"
```

Where node.txt is the file where the IP adreeses of the machines that user wants to agree are specified (an IP by line). "Result" is a folder where vxargs keeps the files that contain the output of each node. The "ssh -x" command is utilized to execute a command in a remote machine, the IP of the remote machine is specified by means of {} and the command is among quotation marks.

Once the vxargs is executed a black screen is shown with the different nodes to the user is being agreed, and the prompt of each node appears. All these outputs are kept in the Result folder, by if the output of each node wants to be analyzed subsequently.

4.5.2. Start application

As the application works on Pastry network is interesting to do that a node be always the bootstrap. So that first a bootstrap node can be launched, for example the node with IP 138.232.66.194:

```
ssh -l upc_epsc -i clave_rsa 138.232.66.194
```

Once we have agreed to future bootstrap node the user must execute the PKI-P2P application:

```
java -cp .:FreePastry-1.4.4.jar:log4j-1.2.14.jar:xmlpull.jar:xpp3-  
1.1.4.zip:bcprov-jdk15-134.jar distapplication/DistPKIDHT 9001  
138.232.66.194 9001
```


To execute the DistPKIDHT class the user must pass him the parameters:

- Local Port: in this case is 9001
- IP Address of the bootstrap node: as there is not bootstrap node we put the IP of the local node
- Port of the bootstrap node

Besides the user must specify the jar files that the application uses:

- FreePastry-1.4.4.jar: where all the functions of Pastry are implemented.
- log4j-1.2.14.jar: where are the functions for the log of java
- xmlpull.jar and xpp3-1.1.4.zip: these two are necessary for the Past operation.
- bcprov-jdk15-134.jar: is utilized for cryptographic algorithms.

Once the bootstrap node is in operation, the PKI-P2P application creates a new Pastry ring. Subsequently the user must execute the other nodes that want to intervene in the test; this is done by means of the vxargs:

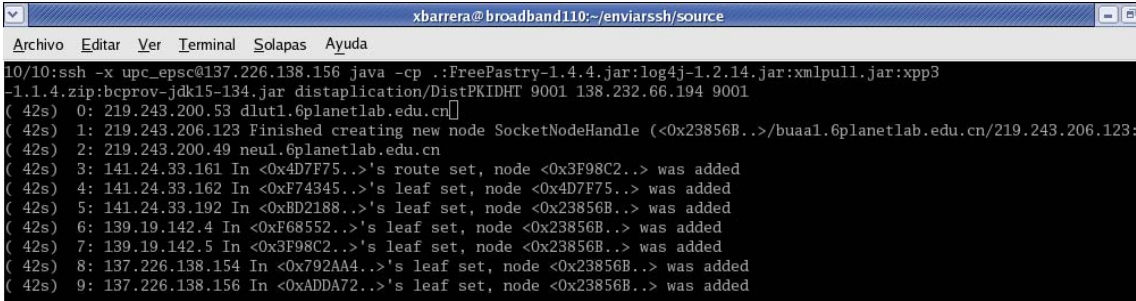
```
python vxargs-0.3.3.py -a node.txt -o /homerresult ssh -x upc_epsc@{}
"java -cp .:FreePastry-1.4.4.jar:log4j-1.2.14.jar:xmlpull.jar:xpp3-
1.1.4.zip:bcprov-jdk15-134.jar distapplication/DistPKIDHT 9001
138.232.66.194 9001"
```

The result is the following one:

```
upc_epsc@plab1-c703:~
Archivo  Editor  Ver  Terminal  Solapas  Ayuda
Connection to 138.232.66.194 closed.
[xbarrera@broadband110 ~]$ ssh -l upc_epsc -i id_rsa 138.232.66.194
Last login: Wed Apr 18 09:44:16 2007 from 147.83.118.72
[upc_epsc@plab1-c703 upc_epsc]$ ps
  PID TTY          TIME CMD
 13980 pts/0    00:00:00 bash
 14197 pts/0    00:00:00 ps
[upc_epsc@plab1-c703 upc_epsc]$ killall java
java: no process killed
[upc_epsc@plab1-c703 upc_epsc]$ java -cp .:FreePastry-1.4.4.jar:log4j-1.2.14.jar:xmlpull.jar:xpp3-1.1.4.zip:bcprov-jdk15-134.jar distapplication/DistPKIDHT 9001 138.232.66.194 9001
:rice.pastry.socket:1176892674647:Error connecting to address /138.232.66.194:9001: java.net.ConnectException: Connection refused
:rice.pastry.socket:1176892674676:No bootstrap node provided, starting a new ring binding to address plab1-c703.uibk.ac.at/138.232.66.194...
Node <0xDC8123..> ready, waking up any clients
Finished creating new node SocketNodeHandle (<0xDC8123..>/plab1-c703.uibk.ac.at/138.232.66.194:9001 [-132858436525321652])
In <0xDC8123..>'s route set, node <0xBD2188..> was added
In <0xDC8123..>'s leaf set, node <0xBD2188..> was added
In <0xDC8123..>'s route set, node <0xF74345..> was added
In <0xDC8123..>'s leaf set, node <0xF74345..> was added
In <0xDC8123..>'s route set, node <0x792AA4..> was added
In <0xDC8123..>'s route set, node <0xADDA72..> was added
In <0xDC8123..>'s leaf set, node <0xADDA72..> was added
In <0xDC8123..>'s leaf set, node <0x792AA4..> was added
In <0xDC8123..>'s route set, node <0x3F98C2..> was added
In <0xDC8123..>'s leaf set, node <0x3F98C2..> was added
In <0xDC8123..>'s leaf set, node <0xF68552..> was added
In <0xDC8123..>'s route set, node <0x4D7F75..> was added
In <0xDC8123..>'s leaf set, node <0x4D7F75..> was added
In <0xDC8123..>'s route set, node <0x23856B..> was added
```

Fig 4.8 Bootstrap output terminal

The nodes contact with the bootstrap by means of the IP address and the port that has been provided them, so that instead of creating a new Pastry ring Pastry the nodes are added to ring created by the bootstrap node. In the previous image can be observed like little by little the bootstrap node goes recognizing the nodes and goes them adding to its leafset and its routing table. This shows that the nodes are communicating through the PlanetLab network.



```

10/10:ssh -x upc_epsc@137.226.138.156 java -cp ./FreePastry-1.4.4.jar:log4j-1.2.14.jar:xmlpull.jar:xpp3
-1.1.4.zip:bcprov-jdk15-134.jar distapplication/DistPKIDHT 9001 138.232.66.194 9001
( 42s) 0: 219.243.200.53 dlut1.6planetlab.edu.cn
( 42s) 1: 219.243.206.123 Finished creating new node SocketNodeHandle (<0x23856B..>/buaa1.6planetlab.edu.cn/219.243.206.123:
( 42s) 2: 219.243.200.49 neul.6planetlab.edu.cn
( 42s) 3: 141.24.33.161 In <0x4D7F75..>'s route set, node <0x3F98C2..> was added
( 42s) 4: 141.24.33.162 In <0xF74345..>'s leaf set, node <0x4D7F75..> was added
( 42s) 5: 141.24.33.192 In <0xBD2188..>'s leaf set, node <0x23856B..> was added
( 42s) 6: 139.19.142.4 In <0xF68552..>'s leaf set, node <0x23856B..> was added
( 42s) 7: 139.19.142.5 In <0x3F98C2..>'s leaf set, node <0x23856B..> was added
( 42s) 8: 137.226.138.154 In <0x792AA4..>'s leaf set, node <0x23856B..> was added
( 42s) 9: 137.226.138.156 In <0xADDA72..>'s leaf set, node <0x23856B..> was added

```

Fig 4.9 vxargs output

As it can be seen in the previous image, all the nodes communicate among them, and they are added each other in the routing tables and leafset. In the vxargs screen the output of line commands of each node can be seen but only the most recent line, so that costs to continue the thread of what this happening in each node, this is an objection of the vxargs. To see the complete output of each node, the vxargs save these outputs as files *.out at final of the session in the folder /result. In the following image can be seen the files *.out generated in one of the tests.

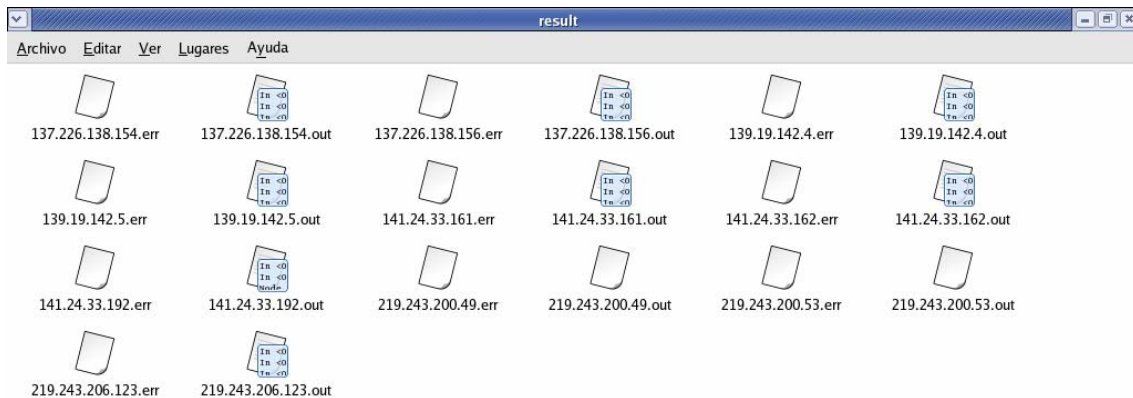


Fig 4.10 Result folder

Also *.err files are generated, these files are destined to communicate to user the failures in the SSH connection. If it has happened no error the file is maintained blank.

4.5.3. Logs

The vxargs provide a log system by means of the *.out files, but If the user not wants to use the vxargs, the PKI-P2P application has an own log system that provides the log4java [11] package.

Inserting log statements into your code is a low-tech method for debugging it. It may also be the only way because debuggers are not always available or applicable. This is often the case for distributed applications.

On the other hand, some people argue that log statements pollute source code and decrease legibility. (We believe that the contrary is true). In the Java language where a preprocessor is not available, log statements increase the size of the code and reduce its speed, even when logging is turned off. Given that a reasonably sized application may contain thousands of log statements, speed is of particular importance.

With log4java it is possible to enable logging at runtime without modifying the application binary. The log4java package is designed so that these statements can remain in shipped code without incurring a heavy performance cost. Logging behavior can be controlled by editing a configuration file, without touching the application binary.

CONCLUSIONS

The work would be able to divide into two large blocks, a first block in which a study of the characteristics of a PKI is done and is implemented in Java on the Pastry P2P network. And another great block in which a study to the PlanetLab is done and as to utilize it to carry out tests with its nodes.

As it has been able to be seen in this memory, an application on FreePastry has been implemented that provides all the functionalities of a PKI: permits the cipher of data achieving thus a sure communication among nodes, also permits the distribution of the certificates of each user among the remainder of nodes, besides each node can sign the certificates that want. Due to the lack of CA's that accredit the relations between public keys and users a public key authentication mechanism has been implemented making use of self-signed certificates and recommendations. The process of authentication has tried to continue as faithfully as possible to Thomas Wölfel document.

The great problem that has arisen in the implementation of PKI-P2P is the distribution of the self-signed certificates and recommendations. To be able to cause to arrive the certificates and recommendations to all the nodes of the network Pastry from distributed form, the best solution passes for utilize a DHT. In FreePastry the only DHT that to have implemented is Past. But the problem is that Past does not fit with the needs that have PKI-P2P. It has two large objections: any object that is inserted in Past cannot be modified and can neither be eliminated. So that if a node obtains a self-signed certificate, trusts in that certificate, and wants to sign it, will not be able to insert again in Past with the same identifier, would have that to insert it as a new Past object with a totally different identifier. This it is the reason by the one that opted for including in Past the self-signed certificates by a band and later by another band each one of the signs that the nodes go generating. All this has complicated enough the development of the application and its operation.

A solution for this problem would be to create an own DHT on FreePastry that to comply strictly with the needs that has PKI-P2P but would be to be deviated a little of the project objectives due to the difficulty that implies the implementation of a DHT since 0.

In conclusion would be able to say that the PKI-P2P application would be able to improve substituting the DHT Past or seeking new solutions, but at present permits that a node authenticates the public key of another node and that the two nodes can maintain an exchange of sure and encrypted data on an insecure channel.

With respect to the part of the PlanetLab, the objective was to evaluate the PlanetLab system to be able to carry out tests with a great number of nodes in a simple way and remotely with SSH since your PC. After knowing a little how functions, and to have done some tests on its nodes would be able to say that if

complies like network of tests but is not what was expected. PlanetLab is a network that at present has 786 nodes; this implies that only tests with that number of nodes can be done. This it is an objection if what intends is to carry out software tests to great scale with 1000 nodes, 10.000 nodes... to emulate a little better the behavior of the application in a network as Internet where there are thousands and thousands of potential users.

Another small objection is the fact that not all the nodes are accessible in a same instant of time; many of them are being reinstalled, or debugging, so that they cannot be utilized. This implies two things, the first one, is that the number of nodes for to do the tests is less than 786, and the other that only the nodes with de boot state can to be used therefore a user must do a prior selection of the nodes. Besides a user must add the nodes one to one by hand to your slice, because only you will be able to agree for SSH to the nodes that have added to your slice. Therefore if a test with many nodes wants to be carried out, if is the first time, turns out to be difficult to have that to add all those nodes one to one since the PlanetLab web.

Another aspect of PlanetLab that does heavy the execution of a test is that the nodes are empty of software, for example they do not have installed the java virtual machine. So that the user should upload all that is needed to carry out the test to each one of the nodes, here is where is necessary to utilize the scripts and the vxargs application to speed up all the process to copy files. In tests of 10 or 20 nodes delays 10 or 20 minutes in having the ready nodes to carry out the test, but for tests with hundreds of nodes the time of wait is extended.

The conclusion is that to carry out tests in PlanetLab implies to dedicate previously a time to add nodes to slice, to prepare those nodes uploading all the necessary files and to leave them ready for the tests. If it will improve in the web the way to add the nodes and all the nodes are 100% functional the service of PlanetLab as tests network would improve a lot.

If tests with a great number of nodes want to be carried out the PlanetLab network is not the solution, but PPlanetLab goes growing day by day incorporating new sites and adding new nodes to service. And at present the PlanetLab software that incorporate the nodes is changing of the version 3 to the version 4 for which perhaps in a future all these small objections are solved.

Future works

An urgent improvement is to improve the adaptation of Past to the PKI-P2P application because its operation doesn't work properly as it has been mentioned before. Also it would be able to find out an alternative to current Past implementation, a DHT on Pastry that adapted better to the needs of PKI-P2P.

The PKI-P2P graphic interface has been implemented quickly, almost at deadline. This is why it must be taken into account to develop a much more user-friendly frontend.

A pending interesting task is to utilize the PKI-P2P application to distribute the user register in SIP applications. The normal thing in SIP is that the users be registered in a well known central server; the idea is to utilize PKI-P2P so that the register information is distributed among all the SIP users instead of its centralized in a server.

Environmental Impact

The fast technological development that has carried out the humans has contributed in a great consumption of the natural resources to a so high rhythm that the planet does not give time to be regenerated. This has caused that in recent years be spoken constantly of the environmental impact caused by the new technologies. Therefore it is very important to carry out an evaluation of the environmental impact of every project be done.

Due to that the project is only developed with software the impact mediambiental is minimum. The only natural resource that has been utilized is the electric power that supply the PC since which develops the project. But on the other hand working with PlanetLab, the expense of electric power already most is raised, because there is almost 800 nodes that are the 24 hours power on.

Personal Conclusions

The theme of this project i have liked because intervene two very important aspects nowadays that are: the P2P networks and their security. The networks P2P are utilized for millions of people everywhere thanks to the success that have had programs as Napster or Emule. At first the security characteristics of in P2P networks were forgot or tried punctual form. Nowadays the P2P reserachers are working in the security in P2P networks; a critical theme for example is the anonymity of the users.

The project has contributed me much knowledge on security in the networks, knowledge that already had studied before but that have been deepened and applied in an environment as critical as are the P2P networks. The part of PlanetLab investigation has also turned out me interesting and has contributed me knowledge of Linux and SSH.

Besides it has motivated me the fact to implement the application on Pastry because is a P2P network that already knew and that I find very interesting by its characteristics and by its routing algorithm.

REFERENCES

- [1] Eckel, B., "Thinking in java"
- [2] Wölfel, T., "Public-Key-Infrastructure Based on a Peer-to-Peer Network"
- [3] **Introduction to Cryptography**,
URL <<http://rinconquevedo.iespana.es/rinconquevedo/criptografia/introduccion.htm>>
May, 18th 2007
- [4] **Introduction to Public Key Infrastructure**
URL <<http://www.mug.org.ar/Infraestructura/ArticInfraestructura/1166.aspx>>
May, 18th 2007
- [5] **Introduction to Cipher Algorithms**,
URL <http://publib.boulder.ibm.com/infocenter/tpfhelp/current/index.jsp?topic=/com.ibm.ztpf-ztpfdf.doc_put.03/gtps5/s5cphr1.html>
May, 18th 2007
- [6] **Introduction to X.509**, URL <<http://es.wikipedia.org/wiki/X.509>>
May, 18th 2007
- [7] **Pastry**, A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001.
- [8] **FreePastry**, URL <<http://freepastry.org/>>
May, 18th 2007
- [9] **Past**, P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility", HotOS VIII, Schloss Elmau, Germany, May 2001.
- [10] **PlanetLab**, URL <<http://www.planet-lab.org>>
May, 18th 2007
- [11] **Log4java**, URL <<http://logging.apache.org/log4j/docs/index.html>>
May, 18th 2007
- [12] **Vxargs**, URL <<http://dharma.cis.upenn.edu/planetlab/vxargs/>>
May, 18th 2007
- [13] **CoDeen**, URL <<http://codeen.cs.princeton.edu/>>
May, 18th 2007
- [14] **Eclipse**, URL <<http://www.eclipse.org/>>
May, 18th 2007

ACRONYMS

ASN.1	Abstract Syntax Notation
CA	Certificate Authority
CN	Common Name
DES	Data Encryption Standard
DHT	Distributed Hash Tables
DSA	Digital Signature Algorithm
HT	Hash Tables
IETF	Internet Engineering Task Force
IP	Internet Protocol
MD5	Message Digest 5
P2P	Peer-to-Peer
PC	Personal Computer
PGP	Pretty Good Privacy
PI	Principal Investigator
PKI	Public Key Infrastructure
RSA	Rivest Shamir Adelman
SHA1	Secure Hash Algorithm 1
SSH	Secure Shell
TCP	Transport Control Protocol
UPC	Polytechnic University of Catalonia

ANNEX A. Test with 50 nodes

ANNEX B. PlanetLab tools

A.1 Introduction

There are applications that improve the experience when working with PlanetLab, they make easier to work with this platform. Some of them are listed below:

- pSSH (Parallel-SSH): Permits to execute SSH commands in parallel up to 30 nodes.
- Vxargs: vxargs is inspired by xargs and pssh. It provides the parallel versions of any arbitrary command, including ssh, rsync, scp, wget, curl, and whatever. One reason to use it is to control a large set of machines in the wide-area network.
- Nixes: This program helps in the installation, maintenance and control of applications in PlanetLab.
- Application Manager: This application is a code that runs on a web-PHP server, and works with PostgreSQL databases.
- LibHttpd++: LibHTTPD can be used to add basic web server capabilities to an application or embedded device. The library handles both static and dynamically generated content, has very low overheads, and provides many features to simplify the creation of web based application interfaces.
- ASD (Asynchron Sensor Daemon): API to know the state of variables in a PlanetLab node.
- SliceTools: set of tools to configure and to manage slices.
- PlanetLab toolkit: Simpler version of the pSSH.

Due to the change of PlanetLab software in the nodes (the version 3.0 to the 4.0), the majority of these tools they are not currently working. Previously in the PlanetLab web there was a 'user tools' section where all these applications were found and many more but due to the change of software this section has disappeared.

The only tools that have been able to be utilized they are vxargs during the tests and the set of tools CoDeen.

B.1 CoDeeN

CoDeeN is an academic testbed Content Distribution Network (CDN) built on top of PlanetLab by the Network Systems Group at Princeton University. This testbed CDN consists of a network of high-performance proxy servers. Currently, proxy servers have been deployed on many PlanetLab nodes. These proxy servers behave both as *request redirectors* and *server surrogates*. They cooperate with each other and collectively provide a *fast* and *robust* web

content delivery service to CoDeeN users. A number of projects are related to CoDeeN, including the following:

- CoBlitz, a scalable Web-based distribution service for large files.
- CoDeploy, an efficient synchronization tool for PlanetLab slices.
- CoDNS, a fast and reliable name lookup service.
- CoTop, a command-line activity monitoring tool for PlanetLab.
- CoMon, a Web-based general node/slice monitor that monitors most PlanetLab nodes.
- CoTest, a login debugging tool that tries to be human-friendly.
- CoViz, a visualization tool graphically displaying PlanetLab activity.

A. CoBlitz

CoBlitz provides a means to scalably serve large files over an HTTP content distribution network. It requires no modification of clients or servers, since all of the necessary support is located on the content distribution network itself. While it is built using the CoDeeN network running on PlanetLab, it does not require you to actively use CoDeeN or to join PlanetLab.

You add the prefix `http://coblitz.codeen.org:3125/` to the URL you want to serve, and CoBlitz does the rest. CoBlitz uses the same underlying infrastructure as CoDeploy, but is being made available for public access. To give a high-level description of how it operates:

- When clients request a large file, they are really contacting a special agent that resides on the CDN node. This agent looks like a standard Web server.
- The agent converts the single request from the client into a stream of requests for smaller pieces (chunks) of the file. These requests are spread, in parallel, to other peer CDN nodes.
- These peers request the chunks from the origin server, using the byte-range support in HTTP. The peers not only send the request back to the original agent, but also cache their chunks.
- The agent reassembles the chunks and sends them back to the client in order, making it appear like one seamless download.

This approach has several benefits:

- As peers join/leave the CDN, only the missing parts of the large file need to be re-requested, instead of doing whole-file caching.
- Large files can be spread across the main memory of many nodes, reducing the memory pressure on any single node, and reducing the number of disk accesses needed to serve the file.
- Since we use HTTP as the underlying protocol, no changes are required to clients or servers. All CoBlitz support is on the CDN itself.

B. CoDeploy

CoDeploy provides a means to efficiently and scalably distribute content from one source to many receivers. In practice, what this means for PlanetLab is that it allows you to push content to hundreds of PlanetLab nodes without having to consume lots of bandwidth at the source. In general, these techniques can be used for efficient peer-to-peer hosting of arbitrary content. CoDeploy uses a number of techniques to perform this distribution efficiently, such as:

- Using HTTP caching via the CoDeeN content distribution network.
- Splitting large files into multiple pieces so that even files that are hundreds of megabytes can be handled efficiently.
- Locating suitable CoDeeN nodes via a "peer review" system.

C. CoDNS

The CoDNS service provides cooperative name lookups to provide a faster, more reliable DNS lookup service. It is a thin wrapper for name lookup which dramatically reduces the client side latency while consuming minimal resources. We have found that nameservers often experience local failures, resulting in clients incurring many seconds of delay, even for cached records. In that case, a typical 50 ms lookup suddenly increases by a factor of 100 or more. CoDNS solves this problem by redirecting the lookup query to a healthy peer node when the local nameserver starts to reveal failures. This masks off the long latency in name lookups caused by local failures and provides consistently fast, reliable response to virtually all name lookup requests.

D. CoTop

CoTop provides a top-like monitoring tool for PlanetLab. What this means is that instead of seeing processes and their properties, you see information about slices. This approach provides a means of seeing what slices are consuming resources on each node, without requiring access to information about all processes.

CoTop is intended to be very similar to the standard top. When you run it, you'll see a summary of the system at the top, followed by a number of rows. On each row, you will see the summary for one slice on the node. These rows are sorted by CPU consumption by default, but the sort key can be changed by pressing numbers in the range of '2'-'0' (all numbers except '1').

E. CoMon

CoMon provides a monitoring statistics for PlanetLab at both a node level and a slice level. It can be used to see what is affecting the performance of nodes, and to examine the resource profiles of individual experiments.

The status page provides several views of PlanetLab, including node-centric, slice-centric, and others. To see more views, click on any of the links shown in the "Summaries:" line at the top of each page. Also available are pages showing the nodes with problems, and the slices with problems, which can be useful for general problem monitoring.

F. CoTest

CoTest is a login debugging tool for PlanetLab. If you are having problems logging into a node, you can run CoTest to see what various data sources think about the node in question. The output is meant to be human-friendly. This tool gets its inspiration from Neil Spring's "why" script.

You download CoTest, compile it, and run it on the command line. You provide it your slice name, and a list of nodes, and for each node, it provides some information about any problems the node is experiencing. It currently pulls data from two sources - the CoMon and CoTop. It uses a fairly simple process to determine what problems might be occurring, and while it's not perfect, it should be reasonably accurate.

G. CoViz

CoViz provides some visualizations of PlanetLab usage. The visualizations show various metrics of PlanetLab activity, and are updated every 5 minutes. The underlying data is taken from the CoMon project.

The goal of CoViz is to be useful to several communities, while being visually interesting as well. All of the data shown in CoViz is meant to be useful for administering PlanetLab -- it quickly shows what experiments may be acting strangely, without providing an overwhelming level of detail. It also presents a sense of proportion, to see how the overall resources are being used. We have found that this kind of "feel" is hard to achieve just by examining the raw data in CoMon. Finally, CoViz is meant to provide some "eye candy" for PlanetLab, since most network monitoring is relatively visually unappealing. CoViz also provides an auto-updating slide show.

CoViz uses the TreeMap visualization format developed by Professor Ben Schneiderman of UMD, and the TreeMap library developed by his research group. In this format, a rectangle is repeatedly subdivided to show individual elements, with the size of each area related to its importance. One fairly popular use of this technique is the "Map of the Market" from SmartMoney. Brent Chun previously used this technique for PlanetLab visualization.

We currently have eight different visualizations, in three categories: resources, efficiency, and usage. The resources category refers to the bandwidth, CPU, and memory used by the various slices. Efficiency refers to how much bandwidth each slice is generating or consuming for the amount of CPU or

memory used. Finally, usage refers to how different sites are using PlanetLab in terms of experiments, nodes, and number of slivers.

Each rectangle contains one label and two values. The label is the name of the slice or site, and the two resources reflect the values used to create the visualization. The first value is what is used to determine the size of each rectangle. The second value is what determines the rectangle's color. The specifics of each visualization are given below, but the general trend is that red is worrisome, black is unsurprising, and green is desirable.

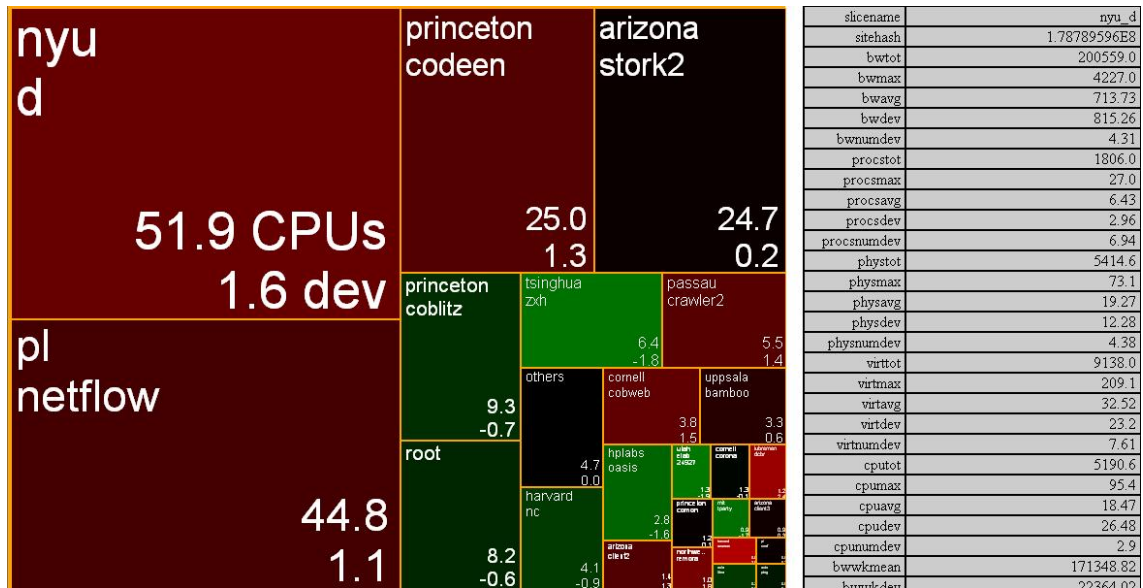


Fig 0.1 CPU Slice visualization

An example of visualization is the previous image. This image shows the CPU consumption of each active slice on PlanetLab. The size of each rectangle reflects the slice's fraction of total CPU consumption on PlanetLab. Slices that consume less than 0.2% of the aggregate CPU are coalesced into a box labeled "Others". The size value is the total aggregate CPU used across PlanetLab, in the unit of CPU powers. For example, if a slice is running on 7 nodes and using 30% of each node, we say that it is using 2.1 total CPUs. The color value reflects how much above (red) or below (green) the current usage is compared to the weekly mean. CoViz calculates the standard deviation of usage over the past week, and use this to select color. Two standard deviations results in a shade of 50% red or green, while four or more deviations results in a 100% red or green. No color is assigned to the "others" rectangle, since we do not keep track of the average of the coalesced slices.

ANNEX C. Scripts

This script installs and configures the java. Parameters: IP address of the node, binary file of java, jar file of FreePastry, jar file of log4java, script that installs the java, and the file for the configuration of Java variables.

```

node="$1"
j2re="$2"
fpastry="$3"
log4="$4"
installjava="$5"
variable="$6"

echo Starting instalation...
echo $node
echo $j2re
echo $fpastry
echo $log4
echo $installjava
echo $variable

echo =====
echo $node
echo Copying J2RE to $node
scp $j2re upc_epsc@$node:$j2re
echo Copying script installjava
scp $installjava upc_epsc@$node:$installjava
echo Copying $fpastry
scp $fpastry upc_epsc@$node:$fpastry
echo Copying $log4
scp $log4 upc_epsc@$node:$log4
echo Copying script java.sh
scp $variable upc_epsc@$node:$variable
echo Entering at $node
ssh -x upc_epsc@$node "echo Installing J2RE at $node; ./$installjava $j2re"
echo =====

```

This script installs the java in the node. Parameters: binary file of java.

```

j2re=$1
echo Installing java.bin
echo "yes" > yes
sh $j2re < yes
rm yes
echo Rename dir jre1.5.0_11/ to java/
mv jre1.5.0_11/ java/
su -c 'cp /home/upc_epsc/java.sh /etc/profile.d'

echo Setting Environments variables
echo "CLASSPATH=.\$HOME/java/lib

```

```

export CLASSPATH

JAVA_HOME=\$HOME/java
export JAVA_HOME

JDK_HOME=\$JAVA_HOME
export JDK_HOME

PATH=.:\$PATH:\$JAVA_HOME/bin
export PATH" >> .bashrc

source .bashrc

```

This script installs and configures the java for several nodes by sequential way.

```

j2re="$1"
fpastry="$2"
log4="$3"
installjava="$4"
variable="$5"

echo Starting instalation...

echo $j2re
echo $fpastry
echo $log4
echo $installjava
echo $variable

for node in 138.232.66.194 200.19.159.35 138.232.66.195 200.19.159.36
do

    echo =====
    echo $node
    echo Copying J2RE to $node
    scp $j2re upc_epsc@$node:$j2re
    echo Copying script installjava
    scp $installjava upc_epsc@$node:$installjava
    echo Copying $fpastry
    scp $fpastry upc_epsc@$node:$fpastry
    echo Copying $log4
    scp $log4 upc_epsc@$node:$log4
    echo Copying script java.sh
    scp $variable upc_epsc@$node:$variable
    echo Entering at $node
    ssh -x upc_epsc@$node "echo Installing J2RE at $node; ./$installjava $j2re"
    echo =====
done

```

This script copies all that is needed to PlanetLab node and executes the PKI-P2P application. Parameters: IP Address of the bootstrap node and the port of the local node.

```
node="$1"
boot="$2"
port="$3"

echo Starting instalation...
echo $node

echo =====
echo $node
ssh -x upc_epsc@$node "rm -R application; rm -R distapplication; rm -R
crypto; rm -R Messaging; rm -R PKIObjects; rm -r privateView"
echo Copiando carpetas to $node
scp -r application upc_epsc@$node:application
scp -r crypto upc_epsc@$node:crypto
scp -r distapplication upc_epsc@$node:distapplication
scp -r Messaging upc_epsc@$node:Messaging
scp -r PKIObjects upc_epsc@$node:PKIObjects
scp -r privateView upc_epsc@$node:privateView
scp bcprov-jdk15-134.jar upc_epsc@$node:bcprov-jdk15-134.jar
scp log4j.properties upc_epsc@$node:log4j.properties
scp xmlpull.jar upc_epsc@$node:xmlpull.jar
scp xpp3-1.1.4.zip upc_epsc@$node:xpp3-1.1.4.zip
echo Ejecutando demopastry en $node con bootsrap $boot
ssh -x upc_epsc@$node "echo ejecutando aplicacion at $node; java -cp
.:FreePastry-1.4.4.jar:log4j-1.2.14.jar:xmlpull.jar:xpp3-1.1.4.zip:bcprov-
jdk15-134.jar distapplication/DistPKIDHT $port $boot 9001"
echo =====
```

This script stops the PKI-P2P application in the PlanetLab nodes.

```
echo Starting pastry...
for node in 138.232.66.194 200.19.159.35
do
    ssh -x upc_epsc@$node "echo cerrando aplicacion at $node; killall java"
done
```


ANNEX D. Log4Java configuraton file

```
log4j.rootCategory=ALL, Default
log4j.appender.Default=org.apache.log4j.FileAppender
log4j.appender.Default.Threshold=INFO
log4j.appender.Default.ImmediateFlush=true
log4j.appender.Default.file=resultado.txt
log4j.appender.Default.layout=org.apache.log4j.PatternLayout
log4j.appender.Default.layout.ConversionPattern=%d %-5p
%C.%M(%L)===> %m%n
log4j.appender.Default.append=false
```