HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Electrical and Communications Engineering
Networking Laboratory

Ignacio González Olías

Security and auto-configuration of Location Servers for IP Telephony

Master's Thesis

Instructor: MSc. Jose Costa Requena

Supervisor: Professor Raimo Kantola

Espoo, March 2002

Abstract

Author	Ignacio González Olías
Title of the Thesis	Security and auto-configuration of Location Servers for IP Telephony
Date	March 2002
Number of pages	113
Faculty	Networking Laboratory Helsinki University of Technology (HUT)
Supervisor	Professor Raimo Kantola
Instructor	MSc. Jose Costa Requena

IP networks were designed to provide an efficient and robust mechanism for transferring data between remote computers. Initially, the information exchanged was mostly computer related data: text documents, files, web content, etc. However, the increasing popularity of this kind of networks is encouraging the development of new services and applications that take advantage of the existing infrastructures.

One of the emerging applications is IP telephony, a new service aimed to allow end users to complete voice calls over IP networks. This a clear example of how a service that was traditionally offered using a different technology, i.e. public switched telephone networks (PSTN), has been adapted to the IP networks.

Since the migration from one approach to another can not take place overnight, both technologies will be interacting for a period of time. This interaction generates new interoperability problems that must be addressed. Data and signalling conversion between both networks is performed by special entities called gateways.

We focus our study on the architecture that solves the problem of gateway location. The Telephony Routing over IP (TRIP) protocol is responsible for distributing gateway availability information between different administrative domains, while the Server Cache Synchronization Protocol (SCSP) performs the required database replication.

This document presents security and auto-configuration modules for the TRIP/SCSP architecture. The former is aimed to ensure that the data exchange takes place in a secure way, while the latter improves the scalability, reliability and availability of the system.

We also include some test results to show how the addition of security services to protect SCSP traffic affects the performance of the protocol.

Keywords: interoperability, gateway location, TRIP, SCSP, replication, security, auto-configuration.

Summary in Spanish

Autor	Ignacio González Olías
Título del proyecto	Security and auto-configuration of Location Servers for IP Telephony
Fecha de lectura	15 de Marzo 2002
Número de páginas	113
Universidad	Helsinki University of Technology (HUT)
Departamento	Department of Electrical and Communications Engineering
Laboratorio	S-38 Networking Laboratory
Supervisor	Professor Raimo Kantola
Instructor	MSc. Jose Costa Requena
Coordinadores	Ángel Álvarez Rodríguez (ETSIT) Anita Bisi (HUT)

Resumen del Proyecto de Fin de Carrera

El imparable crecimiento de las redes IP, en especial Internet, está propiciando la aparición de nuevas aplicaciones que puedan aprovechar las infraestructuras existentes. Dentro de estos servicios destacan aquellos que tradicionalmente han sido ofrecidos mediante el uso de otras tecnologías de red, entre ellos la telefonía.

La telefonía IP permite a los usuarios finales establecer llamadas de voz en redes IP. En una primera fase, esta tecnología está siendo introducida en redes de ámbito privado, permitiendo a las compañías reducir considerablemente los costes de comunicaciones de voz entre diferentes sucursales.

Dado que la telefonía IP conlleva una serie de ventajas y permite un mejor aprovechamiento del ancho de banda, es de esperar que en un futuro no muy lejano se acabe imponiendo sobre la telefonía tradicional. Sin embargo, esta transformación requiere un periodo de adaptación en el que ambas tecnologías deben coexistir. Es precisamente esta fase intermedia la que plantea una serie de problemas en términos de interconexión, ya que en muchos casos el usuario que inicia la llamada y aquel al que va destinada se encuentran en redes diferentes. Gran parte de los esfuerzos actuales en materia de I+D se centran en el desarrollo de mecanismos eficientes que resuelvan estos problemas de interconexión.

Cuando una llamada originada en una red IP tiene como destinatario a un usuario situado en la PSTN, debe existir una entidad intermedia capaz de convertir la señalización y la voz entre ambas tecnologías. Este dispositivo recibe el nombre de gateway, y se caracteriza por tener conectividad IP y por estar también conectado a la red de conmutación de circuitos.

Cada operador dispone de un conjunto de gateways para enrutar llamadas entre las dos redes, lo cual significa que el número de gateways disponibles es muy elevado. Ante esta situación se plantea el problema de cómo elegir un gateway cuando una nueva llamada debe ser establecida. En principio, todo gateway es capaz de enrutar dicha llamada hacia cualquier número de la PSTN, por lo que cualquier gateway de cualquier operador podría ser elegido arbitrariamente para llevar a cabo esta función. Una posible solución sería mantener una base de datos global con información acerca de los gateways. Cuando un usuario desease realizar una llamada hacia la PSTN, símplemente sería necesario enviar una petición al servidor de bases de datos y obtener la dirección de un gateway con las características deseadas.

Sin embargo, existen algunos factores adicionales que imposibilitan el uso de esta solución. Como hemos apuntado anteriormente, cada operador dispone de su propio conjunto de gateways, pero es muy poco probable que quiera que éstos estén disponibles para cualquier usuario final. Cada llamada enrutada a través de un gateway supone un coste para el operador que administra dicho gateway. Lo más probable es que el operador sólo quiera ofrecer sus gateways a sus clientes directos, o a los clientes de otros operadores con los que existe un acuerdo previo de interconexión.

De este modo, la disponibilidad de los gateways debe estar regida por una política local que recoja todos los acuerdos de interconexión entre diferentes operadores. La consecuencia directa es que no todos los clientes finales deben tener acceso al mismo conjunto de gateways. La disponibilidad dependerá del operador que esté ofreciendo servicio al cliente. Esto significa que la idea de mantener una base de datos global debe ser descartada, puesto que choca frontalmente con el concepto de política local.

El protocolo Telephony Routing over IP (TRIP) fue creado por el IETF para tratar de dar solución al problema de cómo encontrar un gateway adecuado para enrutar una llamada. La primera versión de este protocolo se llamaba Gateway Location Protocol (GLP), y utilizaba el Server Cache Synchronization Protocol (SCSP) para replicar la información sobre los gateways. Tras unos años de investigación se decidió incluir el mecanismo de sincronización dentro del propio protocolo, surgiendo así el protocolo TRIP.

Sin embargo, este documento considera una arquitectura intermedia, en el que TRIP se encarga del proceso de toda la información sobre los gateways y SCSP es responsable de la replicación de datos.

El objetivo de este proyecto es analizar la arquitectura TRIP/SCSP y tratar de identificar todas sus debilidades en materia de seguridad y auto-configuración. En principio, SCSP no contempla en su especificación ningún mecanismo de auto-configuración para los servidores, lo cual supone una seria limitación si queremos que nuestra arquitectura sea escalable. El objetivo del módulo de auto-configuración es facilitar la adición de nuevos nodos al sistema sin que sea necesaria ninguna intervención manual. El módulo de

seguridad tiene a su vez como objetivo primordial el garantizar que todo el intercambio de información de routing dentro del sistema tiene lugar de manera segura.

El trabajo realizado en este proyecto parte de una implementación del protocolo SCSP realizada en este mismo laboratorio. Este prototipo incluye soporte SNMP y una implementación de la MIB de SCSP, de manera que los nodos de la arquitectura pueden ser remotamente gestionados utilizando el centro de gestión de red del operador.

Otro de los objetivos de este proyecto es analizar la capacidad en términos de rendimiento del prototipo y diseñar algunas modificaciones que permitan una optimización del mismo.

Preface

This Master's Thesis has been written at the Networking Laboratory of Helsinki University of Technology. The research project belongs to the IMELIO project funded by Tekes, Nokia Networks, Nokia Research Center and Elisa Communications. The thesis has also been financially supported by a grant from the European Union within the Erasmus mobility program.

First of all I would like to thank my supervisor, Professor Raimo Kantola, for giving me the opportunity of joining the lab. Working here has been a great experience and I will never forget it.

I am specially grateful to my instructor Jose Costa Requena for his constant support and friendship. This thesis could not have been written without his valuable guidance and advice. I would also like to give special thanks to Queca, who has been keeping an eye on me all the time.

I sincerely thank all the personnel in the lab, specially my friends Nicklas Beijar, Sampo Kaikkonen, Piia Pulkkinen, Kimmo Pitkäniemi, Jari Huttunen, Juan Francisco Redondo, Juan Ventura, Ramiro Pedros and Tyh-Dar Fan. Special mention to Julio Ramírez, for his help at the beginning of my work, and Moisés Navarro, for being my best friend at the lab and sharing with me all those memorable "Arnolds", "pullas" and "maitos".

During my stay in Finland I have done some good friends. I would like to thank them for those great moments we spent together in Helsinki. Of course I can not forget my friends in Spain. We have strengthened our friendship in spite of the distance and they were always there when I needed their support.

Last but not least, I would like to thank my parents, Paco and Ana, and my sisters, Popo and Lola, for their love and hours of phone calls. I have missed them every single day, and I know they have missed me as well.

This thesis is dedicated to my girlfriend Laura, she knows why. Thanks for bearing with me so long. I love you.

February 20, 2002
Espoo, Finland

Ignacio González Olías

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AES	Advanced Encryption Standard
AH	Authentication Header
AS	Autonomous System
ASN.1	Abstract Syntax Notation One
BGP	Border Gateway Protocol
CA	Cache Alignment
CA	Certificate Authority
CFB	Cipher Feedback Mode
CRL	Certificate Revocation List
CS	Certificate Server
CSA	Cache State Advertisement
CSAS	Cache State Advertisement Summary
DCS	Directly Connected Server
DES	Data Encryption Standard
DN	Distinguished Name
DNS	Domain Name System
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
ESP	Encapsulating Security Payload
FSM	Finite State Machine
GLP	Gateway Location Protocol
GNU	GNU's Not Unix
HFSM	Hello Finite State Machine
HMAC	Hash Message Authentication Code
IANA	Internet Assigned Numbers Authority
ICV	Integrity Check Value
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IP	Internet Protocol
IPTEL	IP Telephony
ITAD	Internet Telephony Administrative Domain
ITU	International Telecommunication Union
LDAP	Lightweight Directory Access Protocol
LNP	Local Number Portability

LS	Location Server
MAC	Message Authentication Code
MIB	Management Information Base
NBMA	Non Broadcast Multiple Access
NIST	National Institute of Standards and Technology
OSPF	Open Shortest Path First
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
PSTN	Public Switched Telephone Network
RA	Registration Authority
SA	Security Associations
SAD	Security Association Database
SAP	Service Access Point
SCSP	Server Cache Synchronization Protocol
SDL	Subgroup Discrete Logarithm
SG	Server Group
SPD	Security Policy Database
SPI	Security Parameter Index
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TRIP	Telephony Routing over IP Protocol
TTL	Time To Live
UDP	User Datagram Protocol
VPN	Virtual Private Network

Chapter 1

Introduction to IP Telephony

This chapter presents a brief introduction about the current state of IP telephony. The introduction discusses the existing proposals aimed to address the problem of gateway location for PSTN-IP interoperability, and describes the TRIP/SCSP architecture.

As IP networks are becoming more popular, new applications are being developed in order to take advantage of the existing infrastructures. IP telephony is one of the new services. This service allows end users to complete telephony calls through an IP network. The final goal is to completely replace conventional telephony by IP based telephony. Since this process requires some time and effort, it has to be carried out in progressive stages. The task is not easy and the architecture needs to be well designed to succeed.

At first, IP telephony has been introduced only in private networks, as the technology needed to expand it into public networks is under development. In a second phase, IP telephony is expected to be gradually connected to the existing PSTN [1]. This means that both technologies will be interacting for a period of time by means of a public interface. The main part of the current research is focused on this area. A decisive target is to develop reliable and optimal mechanisms to make both networks interact.

Several architectures and protocols have been proposed to match the requirements of the new telephony. A comprehensive description of the overall architecture can be found in [2], where the required protocols and their interactions are presented.

1.1 Telephony Routing over IP protocol

The Telephony Routing over IP (TRIP) [3] protocol was specified to address the problem of gateway location. When a telephone call originated in an IP network needs to be routed to a destination located within the PSTN, it

must traverse a device capable of performing data and signaling conversion between different technologies. This device is called a gateway, and it is situated on the boundary between the IP and PSTN networks. A gateway has both IP and circuit switched network connectivity, and plays a central role in IP-PSTN integration. As telephony services over IP networks are becoming more popular, the number of existing gateways is increasing quickly. In addition, different gateways may belong to different operators, and they may have different features and capacities.

In theory, every gateway is capable of routing calls from the IP network to the PSTN. The main problem is how to choose an optimal gateway to route every new IP-side originated call towards the PSTN. The number of possible candidates is large, since every gateway connected to both networks is a potential intermediary to be used in the connection. We must also take into account that a gateway with connectivity to the PSTN is capable to reach every terminal on this network.

One solution to this problem could be the use of a global database to store gateway information. Every time a call needs to be routed towards the PSTN, the required information would be retrieved from the database. This solution assumes that every gateway is made available to everyone, and overlooks the possible existence of different operators running different policies.

However, the expected scenario is likely to be rather complex, with many gateway providers operating at the same time and making their gateways available depending on local policies. This means that one specific provider may be willing to let other providers make use of its resources or not, depending on the policy it is running. Thus, the idea of a global database must be discarded since it does not match the real requirements. Instead of that, every provider will run its own local policy and, as an output of its decision process, will make available some of its resources. This will eventually lead to a situation where databases containing telephony routing information are very different for each provider.

To deal with this scenario, providers can make use of the TRIP protocol to exchange information about gateway availability between different domains.

1.2 IP telephony evolution

The first version of the TRIP protocol was called Gateway Location Protocol (GLP) and is defined in [4]. A description of the framework [5] was proposed as well. That version used the Server Cache Synchronization Protocol (SCSP) [6] to maintain the distributed database of routing information.

After some discussion, the IETF IPTEL working group decided to include the replication mechanism within the protocol itself, creating a new

protocol called TRIP. This new version was modeled after the Border Gateway Protocol (BGP4) [7], and enhanced with some features taken from the Open Shortest Path First (OSPF) protocol [8] and the SCSP.

The architecture that will be discussed in this document is a mixture between the two proposals described. The TRIP protocol is responsible for processing all the routing information, but TRIP messages are not used to transport the information between peers. Instead, the replication task is performed by SCSP for both inter and intra domain relationships.

1.3 The Server Cache Synchronization Protocol

The Server Cache Synchronization Protocol (SCSP) [6] addresses the problem of database synchronization within a set of servers, also called Server Group (SG). The server under scrutiny is called Location Server (LS), and it is connected to one or more Direct Connected Servers (DCS). Any other protocol that requires database replication may use the cache synchronization services provided by SCSP. The protocol has three phases:

- **Hello phase.** During this phase, the LS and its DCSs begin to exchange Hello messages in order to establish a bi-directional peer-to-peer connection. Once this connection has been successfully set up, it must be periodically monitored to ensure that there are no changes in the status of the peer, so Hello messages continue to be exchanged with a predefined frequency. If one node becomes inoperable, its peers will not receive Hello messages from it. In this case, they will close the existing connection and will try to establish a new one.

- **Cache Alignment phase.** Once the connection reaches the bi-directional state, peer servers exchange the contents of their caches to synchronize them. Every server sends a summary of the entire database to its peers so that they can compare it with their own databases. The database entry summaries exchanged are called Cache State Advertisement Summary records (CSAS records). As a result of the comparison, a server decides what information should be requested from each peer. Database entries are exchanged as Cache State Advertisement records (CSA records). When the exchange of the requested data is completed, all the caches within the SG are synchronized.

- **Cache Update phase.** After database synchronization has been achieved, every local change produced in a server cache must be notified to its peers by means of update messages. This phase is responsible for the propagation of updates to keep the caches synchronized so that all the servers have always the same information available. When a server sends an update to a peer, it always waits for the acknowledgment to ensure that the update was properly received and processed.

1.4 The proposed architecture

This section describes the architecture proposed to address the problem of gateway location and presents some definitions about the elements that are present in the TRIP framework. A more detailed description can be found in [9].

All the resources under the control of the same administrative authority form an Internet Telephony Administrative Domain (ITAD). A Location Server (LS) is the entity that uses TRIP to exchange routing information between different ITADs. Every provider is in charge of its own ITAD, and therefore all the LSs belonging to a specific ITAD are driven by the same policy. Thus, all the LSs within the same domain are expected to have the same routing information database.

However, an operator may be interested in having different policies within the same domain. In this case, the operator can divide the ITAD into different areas, so that each area is driven by its own policy. All servers within an area share the same database. The information exchange between areas takes place by means of inter-area connections, which have at least one representative from each area involved in the agreement.

Connections between LSs located in different ITADs are used to exchange routing information between domains. In this case, the connected LSs are called external peers as they exchange inter-domain data. External relationships may involve two or more ITADs. At least one member from each participating ITAD must be present in the relationship.

In order to achieve database synchronization between different servers, this architecture uses SCSP. A SG is used within an area to synchronize all the databases of the LSs belonging to the area. Thus, an ITAD may contain one or more SGs depending on whether it has been divided into areas or not. Additional SGs, called inter-area SGs, are used to replicate databases between members from different areas. Nodes belonging to an inter-area SG must run their decision process to know what routing information they have to propagate to neighboring areas.

Inter-domain relationships use inter-domain SGs to exchange routing information between different ITADs. The overall architecture is illustrated in figure 1.1.

When a node starts running the SCSP daemon, the protocol configuration information is loaded from a file. This information has been determined administratively and includes the set of SGs to which the LS belongs as well as the topology of interconnection within every SG. The file also provides the node with the IP addresses of the DCSs within the SGs so that it can establish a connection to begin the data exchange.

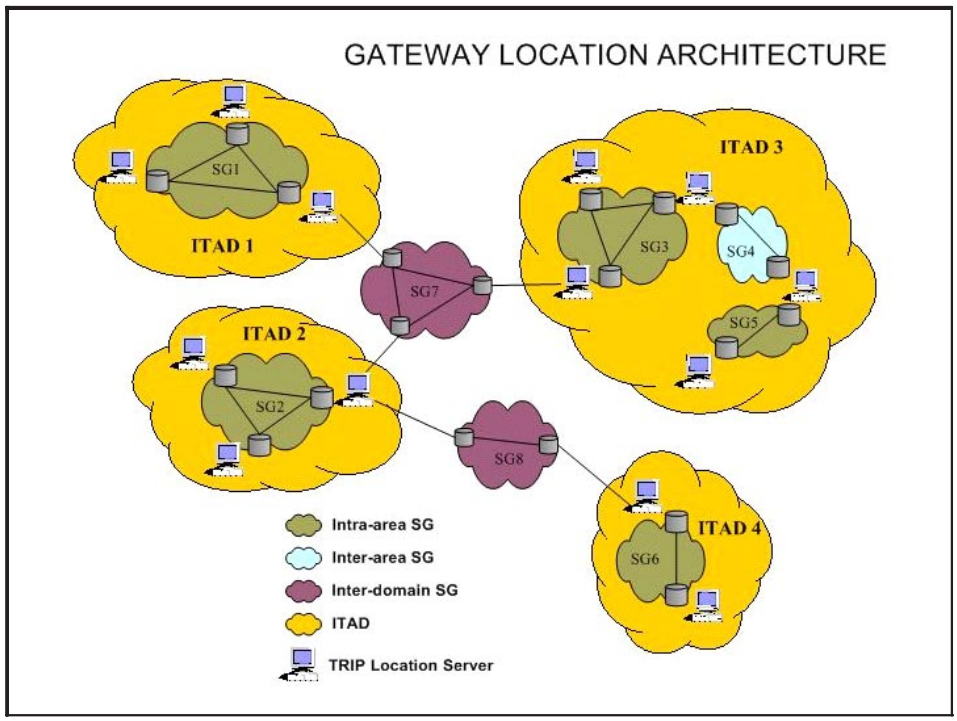

Figure 1.1: Gateway Location Architecture

1.5 Goals of the Thesis

The main goal of this thesis is to develop security and auto-configuration modules for the TRIP/SCSP architecture. These new modules will help to address some limitations identified in the system.

Performance has been a primary concern when developing the SCSP prototype available at the laboratory. Thus, we will try to design and implement the modules in an efficient way. Our goal is to provide the architecture with new functionalities without negatively affecting the performance of the overall system.

Although the modules are planned for a specific SCSP based system, the TRIP/SCSP architecture, our efforts have focused on designing the modules so that they are independent from the application layer. Therefore, they can not only be used within our architecture, where TRIP represents the application layer, but on any other SCSP based system. This allows the modules to be applied to all the existing and future SCSP based systems.

1.6 Structure of the Thesis

The TRIP/SCSP architecture has been presented in this chapter, where we have also described the different elements that take part in the system and their relationships.

The second chapter analyzes the architecture and identifies the main security deficiencies and the drawbacks of the static configuration approach being used so far. Security vulnerabilities are classified into different levels, depending on the architecture level they affect. Thus, the security module can be also divided into different submodules, so that every submodule solves the problems related to a specific security level.

In the third chapter we present a general introduction to cryptography. The main current cryptographic techniques are discussed and their security properties analyzed in order to choose suitable algorithms for our security module.

Chapter fourth describes some performance improvements carried out in the SCSP prototype to enhance the overall operation. Several test results are also presented to measure the performance and characterize the behavior of the new version.

The security module is explained in the fifth chapter. This module consists of several countermeasures aimed to secure the proposed architecture. These countermeasures are based on cryptographic algorithms, which are expected to slow down the speed of the system. Thus, new test results are presented to see how the addition of the module worsens the performance.

In chapter six we describe the auto-configuration module. We first set the requirements for the module, defining those services that are needed to address the static configuration limitations. The rest of the chapter describes the different phases of the proposed auto-configuration procedures.

Chapter 2

Limitations in the architecture

The main goal of this thesis is to propose suitable countermeasures to address some vulnerabilities and limitations that have been detected within the TRIP/SCSP architecture, and that make it not suitable to work in real environments. Therefore, the first step is to identify these limitations in order to achieve a better understanding of the problem.

2.1 Static configuration

As we have already mentioned, each TRIP/SCSP node loads its SCSP configuration parameters from a static file. This means that every single change introduced in the topology of the network must be handled manually by the network administrator. For instance, if one LS is added to an existing ITAD, we must ensure that the SCSP SG corresponding to that domain continues working properly. The configuration files of some of the members of the ITAD must be appropriately updated to guarantee that the routing information is correctly propagated and replicated within the new topology.

Fortunately, SCSP places no topological constraints on a SG, and this feature represents an important advantage over other similar protocols. As an example, BGP, the current exterior routing protocol used for the global Internet, requires fully meshed topologies to interconnect all the members within the same Autonomous System (AS). It means that the addition of a new member to a domain should be notified to all the existing nodes, and this represents a serious scaling problem.

Conversely, SCSP allows database synchronization between nodes interconnected with an arbitrary topology. It is obviously required that the resultant graph representing the SG topology spans the whole set of servers. It is also advisable to provide every node with at least two links with the

rest of the group, so that single points of failure are prevented. This feature provides a great deal of flexibility as well as simplicity in the configuration files of the nodes.

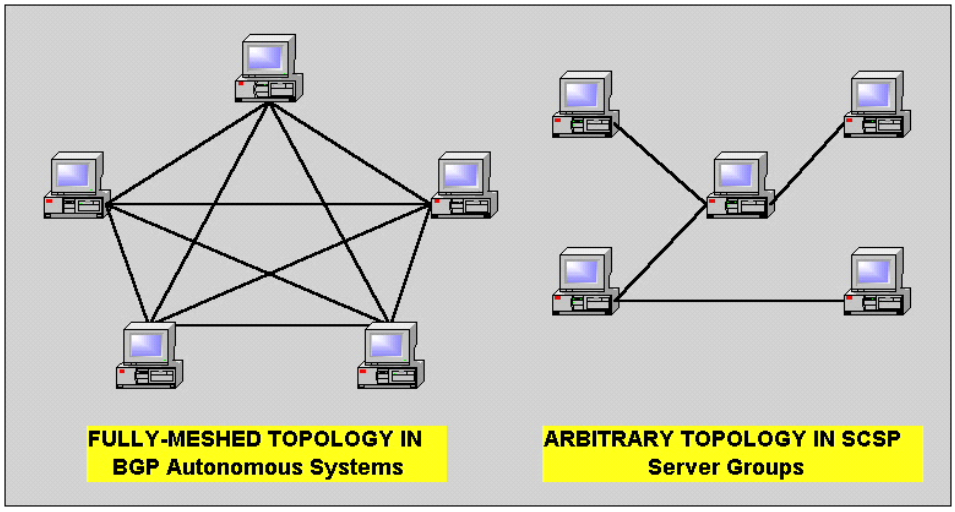

Figure 2.1: Autonomous System topology *vs* Server Group topology

Even though this SCSP feature improves scalability, it is always better to deal with topology changes by means of some automatic mechanism. The advantages of such a mechanism can be summarized as follows:

- Improves the scalability of the system. Manual intervention is no longer required, since all the changes in topology are performed automatically. The automatic mechanism can handle as many changes as needed.
- Prevents the network from accidental misconfiguration errors. Any manual intervention when modifying the configuration files of the servers is subjected to human errors. By eliminating this manual intervention we also improve the reliability of the system.
- Servers do not need to be restarted when there is a change in topology. This results in a higher system availability.

2.2 Security vulnerabilities

The security vulnerabilities identified within the proposed architecture have been classified depending on which architecture level they affect. This approach provides a better comprehension of each vulnerability, and facilitates the task of designing and implementing suitable countermeasures to address

the problems. Table 2.1 shows the security levels of the architecture and the problems that should be solved on each level.

Level	**Security problems**
TCP/IP layer	Vulnerabilities related to the use of the TCP/IP protocol suite in the architecture.
TRIP layer	a)TRIP speaker authentication. b)Routing information integrity over different TRIP hops.
Dynamic Configuration	Security aspects regarding dynamic configuration procedures.

Table 2.1: Summary of security vulnerabilities

2.2.1 TCP/IP related security

The TCP/IP related security lack is due to the use of the TCP/IP protocol suite to carry TRIP messages. TCP/IP protocols were designed almost twenty years ago, when the Internet was still small and reliable, so security issues like encryption and authentication were not a primary concern during the design process. Nowadays, the Internet has evolved to become an insecure network where transactions need to be protected from intruders.

TCP/IP traffic is highly vulnerable to different well-known attacks [10]. One of the most common is IP spoofing, where the intruder pretends to be sending data from an IP address that is actually different from its own address.

Within the TRIP context, IP spoofing allows intruders to capture and modify routing information that is being exchanged between two LSs. In order to ensure the correct operation of the protocol, we should check the following:

- If one TRIP speaker receives some updated information from another TRIP peer, the recipient can be assured that the information was truly generated by the sender, and not modified during the transmission.
- Upon sending one TRIP packet with updated information, the sender can be assured that it was only received by the desired target.
- Even if a third party has access to messages in transit, the sensitive information must remain inaccessible.

If these security requirements are not matched, routing traffic could be vulnerable to several attacks that would eventually lead to an incorrect operation of the protocol. TRIP packets could be intercepted and modified during transmission between peer nodes, and this means a violation of the ITAD

local policies. The final consequence is an undesired operation of the whole IP telephony architecture.

2.2.2 TRIP level related security

TRIP server authentication

Every TRIP server must be provided with some mechanism to verify the identity of its peers. Before setting up the SCSP connections to begin the data exchange, the server must ensure that all its peers are legitimate TRIP speakers and that they have been authorized by their respective ITADs to exchange routing information. This mechanism guarantees the integrity of the routing information being exchanged and prevents the existence of fake TRIP servers. This procedure should be supported by a deployable and reliable authorization mechanism so that TRIP servers can be authorized by their domains to act on behalf of them.

Integrity over TRIP hops

Even if a LS can verify that a received message comes from a determined TRIP speaker, that it has not been modified during transmission and that the speaker is authorized to act on behalf of its ITAD, there still remains the problem of information integrity over different TRIP hops. Updated routing information is generated locally and then made available to neighboring ITADs. An update generated in a specific ITAD may traverse several domains before being received by another ITAD. This means that the update message may have been processed by several LSs during the propagation process. A misconfiguration in any of the traversed LSs would cause the data to be corrupted. Therefore, we conclude that receivers need some mechanism to check the integrity of TRIP route attributes and verify the identity of the LSs that legitimately modified them.

Figure 2.2 illustrates the difference between the TCP/IP related vulnerability and the integrity over TRIP hops security problem.

2.2.3 Dynamic configuration security

The proposed dynamic configuration mechanism must be complemented with some security procedures that allow a node to verify the identity of other peer nodes that want to dynamically join the architecture. The requirements are very similar to those described in the static case, with the difference that now the server is already working.

In the simplest situation, a properly configured SCSP SG is running on an ITAD, and a new node needs to be added to the group. The new member must establish peer to peer connections with one or more servers in the group (depending on the group topology) to begin the database exchange. It runs

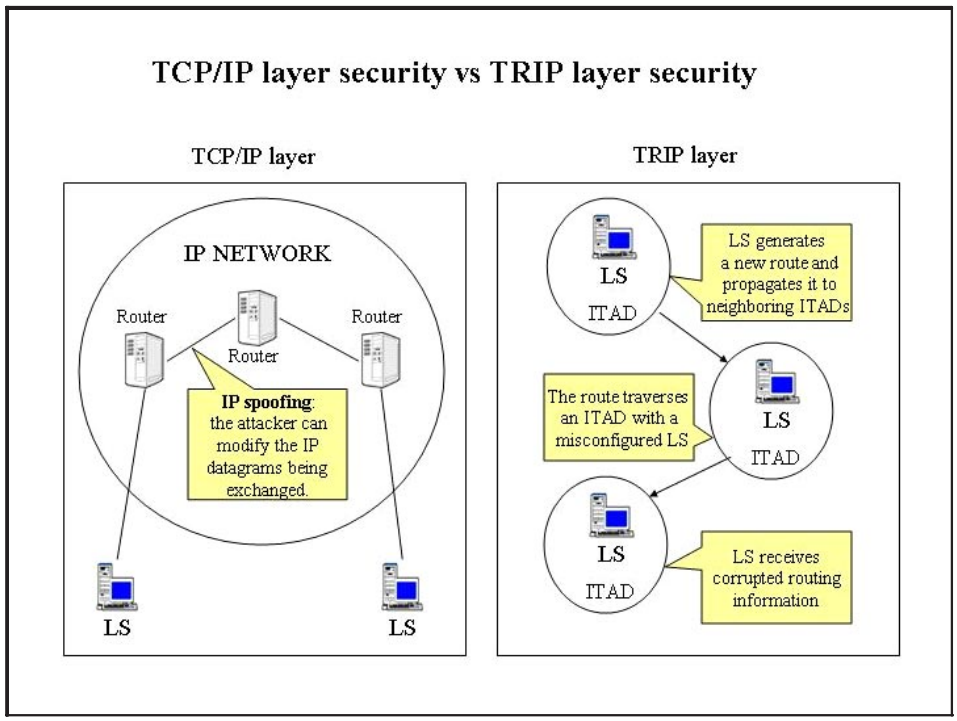

Figure 2.2: TCP/IP related security *vs* TRIP level security

the auto-configuration procedure to obtain the IP addresses of its peers, and starts sending them Hello messages. Upon receiving a Hello message, the peer node checks whether it comes from one of its known peers or not. As the sender is unknown, the recipient must run a checking procedure in order to verify that the received message truly belongs to a legitimate TRIP speaker and that it has been authorized by its ITAD to exchange routing information with other domains.

If this procedure is successful, the new node will be added to the SG. Otherwise, the incoming packet will be discarded. The same description remains valid if the new node tries to set up a connection with a node located in a different ITAD and wants to establish an inter-domain relationship.

Chapter 3

Cryptographic algorithms

This chapter describes the current state of cryptography and presents the most popular algorithms being used. Both symmetric and public key cryptographies are discussed, as well as the main concepts of digital signatures, public key infrastructures and IPSec. In addition, some widespread algorithms are analyzed and compared to find out their security properties and decide whether they are suitable for our purposes or not.

3.1 General cryptographic definitions

If we have some understandable information, and we want to hide it from others, we must apply a mechanism to convert it into unreadable data. The process of transforming an original piece of information, called plaintext, into an unreadable text, called ciphertext, is known as encryption. Decryption is the complementary operation that allows converting ciphertext into plaintext [11].

The science that uses mathematical routines to perform data encryption and decryption is called cryptography. We must not confuse this term with cryptanalysis, which is actually a different science aimed to break secure algorithms.

3.2 Basic security services

There are five basic security services that may be required by an application:

- **Authentication.** Allows an entity to validate the identity of other peer entities within a system.
- **Access control.** Decides whether users are allowed to access the resources of the system, and is typically based on user authentication.

Users are usually given different access permissions according to a security policy.

- **Data integrity.** Ensures that the information has not been modified, and is especially important in data communications over insecure networks, where attackers can easily intercept and modify messages in transit.
- **Data confidentiality.** Provides protection against unauthorized access to private information.
- **Non-repudiation.** Once a transaction between two or more parties has been carried out, this service ensures that none of the participants can later deny that the transaction occurred.

All these basic security services are typically implemented by means of cryptographic algorithms.

3.3 Symmetric cryptography

Symmetric cryptography, also known as private key cryptography, is based on a secret shared key that is used both to encrypt and decrypt messages. When the sender and the recipient of a communication want to exchange messages in a secure way, the former must encrypt the information with the secret key using some predefined algorithm. Upon receiving the message, the recipient uses the same key to decrypt the information.

Symmetric cryptography has been widely used for many years, as it provides a simple mechanism to make a communication secure. Its main advantage is the high speed it achieves in the process of encrypting and decrypting information, but it has important drawbacks that have caused it to be replaced by public key cryptography. The most remarkable limitation of this scheme lies in the fact that the sender and the recipient must agree in using the same secret key before starting to exchange information. If we assume that they are geographically dispersed, this means that the key has to be sent over an insecure channel, so it can easily be intercepted by third parties.

Due to this problem, also called the "key distribution problem", symmetric cryptography is mainly used only in those applications where no key transmission is needed. Actually, as we will discuss later, many current systems take advantage of both public and private key cryptographies by using a combination of them that is known as "hybrid cryptography".

3.3.1 Symmetric ciphers

A cipher is a cryptographic system that converts units of plaintext into units of ciphertext.

Although many different symmetric ciphers have been proposed and implemented, all of them can be classified into two main types: block ciphers and stream ciphers. Block ciphers are the most commonly used and can be easily implemented in software. Encryption is performed by converting fixed-length blocks of plaintext into blocks of ciphertext of the same size.

Stream ciphers are usually faster than block ones, and they are more suitable to be implemented in hardware. Encryption is performed one bit or one byte at a time. Both block and stream ciphers are extensively described in [12].

Block ciphers can operate at several modes, depending on the technique used to encrypt consecutive blocks of plaintext. There are also some additional modes to let block ciphers emulate the operation of stream ciphers. The most common is the Cipher Feedback Mode (CFB), that allows encryption of data units smaller (usually one byte) than the actual block size. This mode is especially useful when block ciphers are used to encrypt arbitrarily sized messages without padding.

Examples of widespread block ciphers are DES, IDEA, SAFER, Blowfish and Rijndael. The latter is used within the Advanced Encryption Standard (AES). The most known stream ciphers are RC4, SEAL and WAKE. Detailed descriptions of all these algorithms can be found in [11] and [12].

The Data Encryption Standard

The Data Encryption Standard (DES), described in [12], is by far the most widespread symmetric algorithm nowadays. It was initially derived from the algorithm LUCIFER, developed by IBM in the early seventies, and has been a worldwide standard for more than 20 years[1].

DES is a block cipher and operates by encrypting blocks of 64 bits. The key length is 56 bits, but eight additional bits are added with parity purposes. The security of the whole system lies on the security of the secret key.

From a theoretical point of view, DES can be still considered a strong symmetric algorithm, since the best known form to attack it is the brute force attack. This attack consists of trying every possible key. For 56-bit keys, this means 2^{56} different possibilities, a number big enough to be completely unfeasible 20 years ago. However, with the growth in power of computer systems, DES has become a rather insecure algorithm, so its use is not recommended.

The Advanced Encryption Standard

In 1997, the U.S. National Institute of Standards and Technology (NIST) announced the requirements for a new encryption standard called Advanced

[1] In 1976 DES was officially adopted by the US government as a federal standard.

Encryption Standard (AES). The Rijndael algorithm, designed by J.Daemen and V.Rijmen, was selected to become the AES. It is a symmetric block cipher that can operate with different block and key sizes. However, the NIST standard adopted a fixed block length of 128 bits, and three possible key lengths: 128, 192 and 256 bits.

The Rijndael algorithm has proven to be extremely resistant to linear and differential cryptanalysis, and its authors claim that the existence of weak keys is highly improbable. The most effective known attack is the exhaustive search (brute force), but in this case the keys used are long enough to ensure the security of the system for many years. The algorithm is extensively described in [11].

3.4 Asymmetric cryptography

Asymmetric cryptography, also called public key cryptography, was invented in 1975 by Whitfield Diffie and Martin Hellman. In this new scheme each party involved in a communication owns two different keys: a public key and a private key. The former is made public and accessible to everyone, whereas the latter is kept secret. Every message encrypted with the public key can only be decrypted by using the corresponding private key and vice versa. However, this mechanism would not be secure if anyone could calculate private keys from public keys. It has been proved that, although both keys are mathematically related, it is computationally infeasible to calculate one of them from the other one, so this scheme is secure enough to match the requirements of most applications.

The most important advantage of public key cryptography is that we eliminate the need to send keys over insecure channels. Public keys are published so that they can be accessed by anyone, and this does not compromise the security of the overall system.

Depending on which key is used to encrypt or decrypt the information, we have different applications of asymmetric cryptography. In the most common case, the sender encrypts the message with the public key of the recipient, which can be retrieved by anyone from a key server. The encrypted message is then sent to the receiver, who uses the corresponding private key to decrypt it. By operating this way, the sender can be assured that the information will only be decrypted by the owner of the true private key. Even though it is possible for an intruder to get a copy of the message, it will not be able to decrypt it, as the private key needed is kept secret by the desired receiver.

The main disadvantage of asymmetric cryptography is that it takes much longer to encrypt and decrypt a message than in the symmetric case. In addition, the key length must be larger in order to achieve a similar level of security performance. For instance, symmetric keys are usually 128 bits

long, whereas it is highly recommended that keys used with asymmetric algorithms are at least 1024 bits long. Moreover, a 128-bit key used in symmetric cryptography has approximately the same security strength than a 3000-bit key used with asymmetric algorithms.

3.4.1 Public-key algorithms

Many public-key algorithms have been proposed since asymmetric cryptography was invented by Diffie and Hellman in the mid seventies. All these algorithms share the main features of public keys schemes: a pair of keys is used to encrypt and decrypt the plaintext and it is computationally unfeasible to derive one key from the other. However, most of the existing algorithms are impractical to be used in real environments. This is due either to the slowness of the computation or to the excessive size of the cyphertext generated. In other cases the problem lies on the lack of security, as the algorithms have been repeatedly attacked and broken by cryptanalysts.

We will focus our study on the RSA and DSA algorithms, which are the most widespread nowadays.

The RSA algorithm

The RSA algorithm [12] was created by R.Rivest, A. Shamir and L.Adleman at the Massachusetts Institute of Technology (MIT) in 1977. Since it is regarded as the easiest to implement and understand, it has been extensively used in many applications[2]. So far nobody has been able to prove whether the algorithm is secure or not. However, since it has resisted many years of cryptoanalysis, it is considered to be secure enough. The security strength of the RSA algorithm is based on the difficulty of factoring large numbers.

The Digital Signature Algorithm

The Digital Signature Algorithm (DSA) [12] was proposed in 1991 by the National Institute of Standards and Technology (NIST) as part of the Digital Signature Standard (DSS). This standard can only be used for digital signatures, which means that it is not suitable for conventional encryption.

Although the time needed for signature generation is very similar in RSA and DSA, the latter performs considerably slower (from 10 to 40 times) in signature verification. On the other hand, DSA has proven to be faster in key generation, but this is not a crucial operation since it only takes place once and it does not concern the final user.

[2]RSA was patented in the United States (not in any other country), but the patent expired on September 20th 2000, so it can now be freely used worldwide.

Security comparison between RSA and DSA

This section is aimed to carry out a security strength comparison between the two public-key algorithms under study. One of the most important parameters we must choose when designing a cryptographic application is the key length. Many recommendations about this parameter are given throughout the cryptographic literature, but they are usually based on assumptions which were only applicable at the time they were written. The security of all public-key algorithms is based on some difficult computational problem, and the private key can only be computed from the public one by solving this problem. Thus, we must always be aware of the current methods available to deal with these computational problems. In general, key length should be carefully chosen regarding two factors: the lifetime of the system and the state-of-the-art factoring technology.

The security strength of the RSA algorithm is based on the difficulty of factoring large numbers, so dramatic advances in this field would compromise the security of the system and would make it vulnerable to attacks.

Factoring algorithms have been slowly improved during the last decades, and many developments have contributed to reduce the difficulty of this task. Therefore, we can assume that this trend will be the same in the future. The new small improvements will be addressed by small increments in key lengths. Obviously, if a revolution in factoring algorithms takes place in the years to come, all our assumptions will have to be revised in order to match the new requirements.

Table 3.1 shows the recommended RSA key sizes for different system lifetimes. These values have been taken from the key size study in [13]. The year column represents the last year in the lifetime of our cryptographic application, which means that our system must remain secure until that year. To calculate these values, the study considers the expected development in computer processing power as well as the forecast cryptanalytic progress.

We can conclude that an approximate size of 1100 bits is reasonably secure for most current applications, but larger values would allow the system to remain secure for decades. On the other hand, encryption and decryption speeds are strongly related to the key length, which means that sizes larger than 1536 bits perform rather slowly in a normal computer. We can thus summarize that there should be a compromise between security and performance requirements when choosing the appropriate key length for the system.

The DSA algorithm is a Subgroup Discrete Logarithm (SDL) system, which means that its security lies on the difficulty of computing discrete logarithms in certain subgroups of a finite field. We can summarize from [13] that the security of a SDL system depends on three parameters: subgroup size, field size and length of the digests generated by the hash function used. Table 3.1 shows the minimum required values for these parameters in order

Last lifetime year	**RSA key size (bits)**	**SDL subgroup size (bits)**	**Hash digest size (bits)**	**SDL field size**
2001	990	126	142	990
2002	1028	127	144	1028
2003	1068	129	146	1068
2004	1108	130	146	1108
2005	1149	131	148	1149
2006	1191	133	150	1191
2007	1235	134	152	1235
2008	1279	135	152	1279
2009	1323	137	154	1323
2010	1369	138	156	1369
2013	1513	142	160	1513
2026	2236	160	180	2236

Table 3.1: Security parameters and system lifetime

to ensure the security of the cryptographic system until the year specified in the first column. The assumptions made to calculate the values in the table are described in [13]. DSA was designed with the following values:

- Subgroup size : 160 bits.
- Field size : From 512 to 1024 bits.
- Hash function digest size : The DSS specification recommends the use of SHA, with 160-bit outputs.

By checking table 3.1 we can see that SDL field sizes of 1024 bits can only be considered secure until year 2002. For the hash digest size and the subgroup size the last valid year would be 2013 and 2026 respectively. The lower bound is thus set by the field size, so we can conclude that the use of the DSS standard may not be adequate beyond the year 2002.

3.5 Digital signatures

So far we have used public keys to encrypt messages and private keys to decrypt them. A different application of public key cryptography arises when the private key is used to sign messages, and the receiver uses the corresponding public key to decrypt the information. This new scheme allows the sender of a piece of information to implement a "digital signature", and its goal is very similar to the one that can be achieved by the use of "conventional signatures":

- **Authentication.** If the receiver succeeds in decrypting the information with a determined public key, then it can be assured that the message was signed by the owner of the corresponding private key. There is still the problem of checking whether the public key truly belongs to the real sender, but this can be solved by means of digital certificates, which will be explained later.
- **Integrity.** If a signed message changes during the transmission, the receiver will not be able to decrypt it with the public key. This is an easy way to detect both accidental and intentional changes in the transmitted information.
- **Non-repudiation.** In some electronic transactions it is important for one of the parties to make sure that the other party can not deny having performed some action. Digital signatures also match this requirement, as an electronic document can only have been signed by the owner of the private key.

3.5.1 Hash functions

Messages generated by a normal application are usually too long to be signed using public key cryptography, as it would take so long to encrypt the information. Instead of signing the whole message, the sender uses a hash algorithm to compute a message digest, which is in fact a compressed version of the original data, and then signs this digest with the private key.

Message digests computed with a specific hash algorithm have always a fixed length (usually 128 or 160 bits), and this length does not depend on the length of the message to be signed. In addition, it is not possible to find out the original message from a digest, and the slightest change in the data to be signed makes the algorithm generate a completely different output value.

A well-designed hash algorithm should be able to resist both a brute force attack and a "birthday attack"[3]. Outputs of 64 bits are considered too small to resist birthday attacks, since they only require hashing 2^{32} random messages to find two with the same resulting digest with a probability of 0,5. To ensure robustness against birthday attacks, outputs of at least 128 bits are strongly recommended. This length makes the attack to be unfeasible given today's technology. However, the evolution in computer's power implies that longer digests will be needed in the future in order to guarantee an acceptable security level.

The choice of an appropriate hash function is an essential decision when we are designing a public key infrastructure, because the use of an insecure algorithm could compromise the security of the whole cryptographic system.

[3]This attack tries to find two arbitrary messages with the same hash value, and is computationally more feasible than the brute force attack.

The most popular hash functions are MD2 [14], MD4 [15], MD5 [16], SHA[4], RIPEMD-160 and HAVAL. They are described in [12], [17] and [18].

Figure 3.1 shows a performance comparison between different hashing algorithms. We have carried out this test using the algorithm implementations available on the OpenSSL 0.9.6b library, available at [19]. Results show that all the tested functions perform acceptably well in software. Moreover, they all have a linear response to the growth in size of the messages to be signed. Results illustrate that the choice of a hashing algorithm for a cryptographic system is not a crucial decision in terms of performance. The SHA family is widely regarded as the most secure, so its use is recommended.

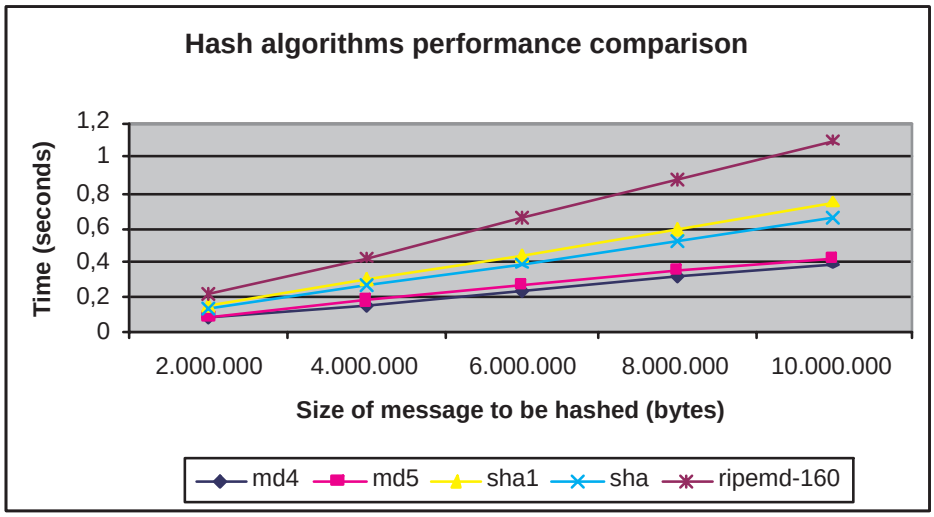

Figure 3.1: Hash function performance comparison

3.5.2 Signing procedure

The required steps to sign and verify a piece of information are described in Figure 3.2. The essential components of such a system are the hashing and public-key algorithms, which should be carefully chosen according to our requirements. The original data is first hashed using the one-way hash function and then encrypted with the private key of the sender. As we explained before, this procedure is aimed to reduce the encryption time, since asymmetric algorithms perform considerably slower than symmetric ones. Both the original data and the signature are sent to the potentially insecure channel.

To validate the integrity of the data, the receiver extracts the sender's public key from its digital certificate and uses it to decrypt the digital sig-

[4]The current version is SHA-1.

nature. The same hash function used by the sender is then applied to the original data, and the resulting digests are compared in order to verify that the message was not modified during the transmission and that it truly belongs to the pretended sender. Notice that the receiver must retrieve the sender's digital certificate from a certificate server, and that this step usually implies the verification of a certificate chain.

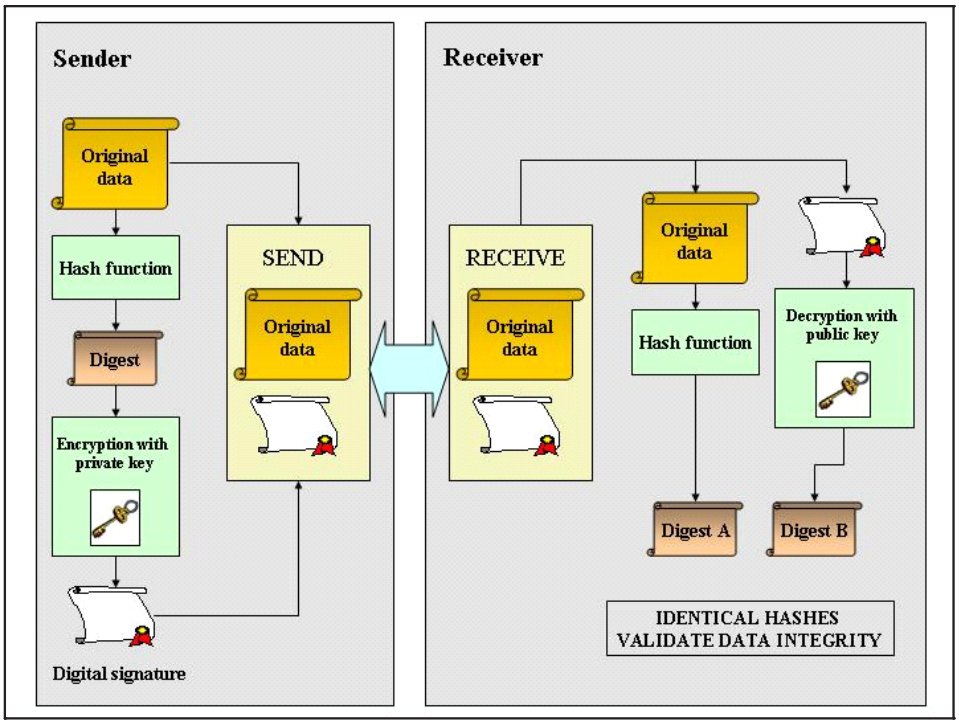

Figure 3.2: Digital signature and verification procedures

3.5.3 The Man-in-the-middle attack

Digital signatures address by themselves the problem of integrity when the information is sent over unreliable networks, but they require some additional mechanisms to ensure that the authentication and non-repudiation properties are matched as well. Upon receiving a digital signature, the receiver entity must ensure that the public key used in the validation process really belongs to the purported signer. Otherwise the communication is highly vulnerable to the "man in the middle" attack.

When two entities (sender and receiver) want to securely communicate using digital signatures, the sender must first send its public key to the receiver to ensure that its own signed messages can be decrypted. If this key exchange takes place over an insecure channel, an intruder can easily

intercept the public key and perform the following actions in order to get access to the confidential information which is being transmitted:

- The intruder generates its own pair of keys.
- The intruder's public key is sent to the receiver instead of the real sender's public key, which is kept to decrypt messages.
- Every sent message is intercepted by the intruder, who uses the sender's public key to decrypt them. The resulting plain text can be modified and re-signed using the fake private key, and sent to the receiver.
- The receiver decrypts the message with the fake public key. The intruder has been able to intercept and modify the communication and neither sender nor receiver have noticed the fraud.

3.5.4 Digital certificates

We can conclude that the proper verification of a digital signature implies that the verifier must have assurance that the signer's public key truly corresponds to its private key. In a public-key cryptographic system, this task is performed by means of digital public key certificates. Digital certificates are electronic documents used to associate the identity of an entity (person, server, company) with a public key, and they are aimed to automate the process of distributing public keys over the network. A certificate basically consists of three sections:

- Information about the entity being verified.
- A public key.
- One or more digital signatures.

Although the user information attached to a digital certificate slightly varies depending on the standard being used, it usually includes the name of the entity, an expiration date, the name of the organization that issued the certificate and a serial number. The certificate bounds this user information to a public key value. Upon receiving a digital signature, the verifier retrieves the signer's digital certificate from a certificate server and uses the attached public key to process the signature.

This mechanism, however, implies that the verifier firmly relies on the contents of the certificate, assuming that the entity described in the document is the owner of the public key. This assumption could not be valid in an insecure environment, where an attacker could have access to the contents of the certificate and freely modify them.

To address this drawback, digital certificates contain a digital signature from a trusted third party authority, called Certificate Authority (CA),

which is responsible for the validation process. The purpose of the digital signature is to assure that the certificate information has been verified by the CA. These authorities are trustable entities that validate identities and issue certificates. Once they have checked that the information contained in the certificate truly belongs to the pretended owner, the CA digitally signs the certificate using its own private key.

Whenever a third party needs to communicate securely with the subject certified by the CA, it retrieves the certificate from a public server and verifies it by means of the CA's public key, which is widely known. If the third party trusts the CA and the verification process is successful, it means that the owner of the public key matches the identity stored in the certificate. With this procedure we can address the problems of authentication and non-repudiation in digital signatures, preventing the use of fake public keys.

3.6 Hybrid cryptography

As we have previously mentioned, most modern systems use a combination of both symmetric and asymmetric algorithms to optimize the performance of the encryption-decryption process. The sender first generates a random session key. This key is used to encrypt all the outgoing traffic using a fast symmetric algorithm. The session key is then encrypted with the recipient's public key and sent as well. Upon receiving the encrypted session key, the recipient makes use of its private key to decrypt it. Once the session key has been decrypted, it can be used to decrypt all the secured traffic.

This approach obtains the benefits of the two schemes already presented: improves the speed of the asymmetric one and avoids the transmission of plain secret keys over unreliable channels. As an example, hybrid cryptography is used by the Pretty Good Privacy (PGP) encryption/authentication program, one of the most popular and widespread security applications [20].

3.7 Public key infrastructures

Certificate management is a trivial task when the number of entities that wish to communicate securely is small. A supporting infrastructure is not needed in that case, since certificate exchange can be manually handled. However, this approach is not practical in an Internet-based environment, where the number of entities taking part in the communication is quickly growing and changing every day. Public Key Infrastructures (PKI) are aimed to solve this scalability problem by providing certificate storage and additional management facilities.

The main components of a basic PKI can be described as follows [21]:

- **Certificate Authority (CA).** As we explained before, a CA is the organization responsible for validating and issuing certificates. The

interface between a CA and its users is provided by a Registration Authority (RA), that authenticates the identity of users and submits the corresponding certificate requests to the CA. The quality of this authentication process usually determines the level of reliability of the PKI. The CA signs the requests with its private key and stamps them with the expiration date.

- **Certificate Server (CS).** A database that can be accessed by users to store and retrieve certificates. It usually includes some additional features to facilitate the maintenance of security policies. The most common implementation is by means of a directory service.

- **Certificate Revocation List (CRL).** Digital certificates have a restricted lifetime which is determined by its start and expiration dates and times, so that it can be considered reliable only within the validity period. Once this time interval has expired, the authenticity of the pair of keys can no longer be assured.

  Furthermore, there are some specific situations in which a certificate must be invalidated before the expiration date. The most common is that in which the certificate holder ends the contract with its company, but it can also happen when the security of the private key has been compromised.

  CRLs are data structures published by CAs that contain a time-stamped list of revoked certificates. This list must be always checked before verifying any digital signature in order to find out whether a specific certificate is still valid or not, and prevents users from using compromised certificates. Revoked certificates are kept in the list until they expire.

3.7.1 PKI standards

The most important organization in PKI standardization is the PKIX working group of the Internet Engineering Task Force (IETF). Its work is focused on interoperability issues to allow different applications accessing a PKI through a uniform interface. In fact, this working group bases its development on two different standards:

- X.509 recommendation from the International Telecommunication Union (ITU). Specifies the authentication service for X.500 directories, described in the X.500 standard, and the syntax of the X.509 certificate format.

- Public Key Cryptography Standards from RSA Security. Developed by RSA Laboratories, provide information about procedures and message formats.

3.7.2 CA hierarchies

A PKI may be composed of one or more CAs depending on the size of the system that is being made secure. For small systems, one CA is responsible for the whole validation process, dealing with all the procedures involved in PKI management: certificate requests, validation, issues, elaboration of CRLs, etc. However, this approach can not maintain a good level of quality validation when the number of users of the PKI grows.

Large PKIs are usually made up of several CAs, so that certificate management tasks can be shared between different validators. The way in which these CAs interact with each other is known as trust model, and the most common one is called top-down hierarchy, where CAs are arranged as a trust tree. This trust model is depicted in figure 3.3.

The CA on top of the tree, known as root CA, is the source of all certification paths and has the ability to certify other CAs. Every CA in the tree can certify their own children, but the last level authorities are exclusively used to certify final users. The number of CA levels used depends on the complexity and structure of the architecture. Every CA gets a signed

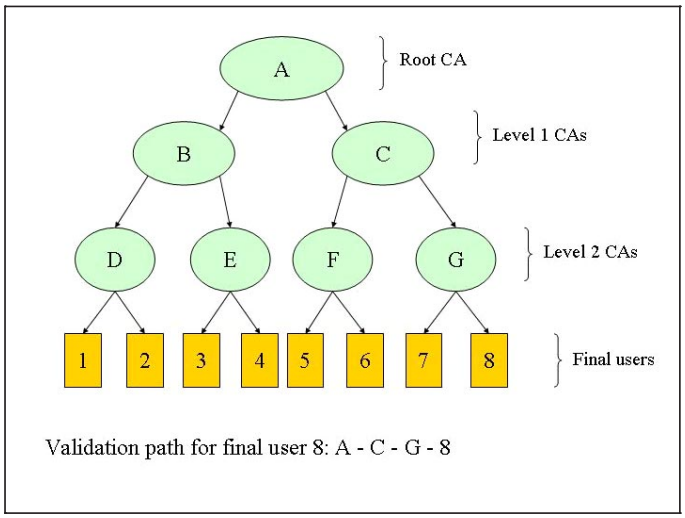

Figure 3.3: Hierarchical PKI structure

certificate from their parent, enabling it to issue certificates to other low level entities, either other CAs or final users. Signature checking requires that the verifier completely relies on the root CA, that owns a self-signed certificate. The verification path starts at this root CA, and goes through the whole tree until it reaches the final user certificate.

As an example, if an entity needs to authenticate the identity of final user 8 in the architecture shown in figure 3.3, it must retrieve the root CA's public key and use it to check the digital signature attached to C's

certificate. Every certificate is then verified using the public key found in the previous certificate. The final user's public key can be obtained by using G's public key to verify the digital signature present in its certificate. In this case, there are three CA levels including the root level, which means that the certificate path involves three certificate validations.

Top-down hierarchies are especially useful in PKIs that operate globally, where final users may be geographically dispersed. They also scale well, since certification paths can be easily discovered. The main disadvantage is that all final users must fully trust the root CA, and they have to obtain a copy of its public key to properly use the PKI.

3.8 X.509 certificates

The most widespread certificate format is the X.509 standard, described in [22], which complies with the ITU-T X.509 international standard. An X.509 certificate consists of a set of standard fields with information about the certified entity. Table 3.2 describes these fields for the X.509v3 specification, which is the current version in use.

3.8.1 X.509v3 certificate extensions

The X.509v3 certificate extensions, defined in [22], provide methods for associating additional attributes with users or public keys and for managing the certification hierarchy. They can also be used by communities to specify private extensions to carry information unique to those communities.

An extension can be defined as critical or non-critical, depending on the procedure followed when a system can not recognize it. A critical extension must be rejected if it can not be recognized; however, a non-critical one may be ignored in such a case.

3.9 IPSec

The IPSec suite is a set of security protocols designed to address the lack of security of the IP layer. In the beginning, the TCP/IP stack was mostly used in academic environments, where designers were not concerned about security issues. However, the significant growth of the Internet implied the creation of new services with strong security requirements, such as banks and e-commerce applications.

IPSec provides the IP layer in both IPv4 and IPv6 with a set of extensively configurable security services which can be used to protect the flow of IP packets between participating IPSec peers. These services include data confidentiality and integrity, access control, data origin authentication and

Field	Description
Version number	Version number of the standard that applies to this certificate.
Serial number	Certificate number assigned by the issuer CA. CAs are responsible for assigning unique serial numbers to the certificates they issue.
User's public key	Includes both the public key itself and a description of the cryptographic algorithms used to generate it.
DN of the certificate subject	A Distinguished Name (DN) is a set of name-value pairs used to uniquely identify the certificate subject. DNs are aimed to uniquely identify entities across the whole Internet. They consist of the following sections: uid : user ID e : e-mail address cn : user's common name o : organization ou : organizational unit c : country
DN of the issuer CA	Uniquely identifies the CA that issued the certificate.
Validity period	Start and expiration date/time.
Signature algorithm identifier	Cryptographic algorithm used by the CA to sign the certificate.
Digital signature	CA's digital signature.

Table 3.2: X.509v3 certificate format

anti-replay mechanisms, and are provided at the IP layer, thus being able to offer protection for both IP and upper layer protocols.

Since the IPSec mechanisms are algorithm independent, they can be implemented using a wide range of different cryptographic algorithms. This implies that the security afforded by the use of the suite services in a system does not only depend on the correct deployment of the architecture, but also on the security strength of the algorithms being used.

The IPSec architecture, its components and their inter-relationships are presented in [23].

3.9.1 Security associations and modes of operation

Security Associations (SA) are a key concept in IPSec and basically represent simplex connections between nodes. SA dynamic establishment and management are crucial tasks for the architecture and are performed by the

Internet Key Exchange (IKE) protocol [24].

The goal of a SA is to secure all the traffic carried by it. Since they are unidirectional by definition, the security of a bi-directional data stream between two hosts always involves at least the establishment of two SAs, one in each direction, although this number may be higher depending on the security services used in the communication.

The IPSec architecture distinguishes between two types of participants: hosts and security gateways. The latter can be regarded as intermediate points in a communication between hosts. The secured traffic is actually destined to a final host, but the topology of the system is arranged so that this traffic must traverse the gateway before reaching its destination. Thus, security gateways are typically installed in Internet routers or firewalls.

There are two different modes of operation depending on the type of the entities involved in the communication. The transport mode is mainly used for host to host security associations, where the ends of the communication actually coincide with the entities speaking IPSec. In contrast, tunnel mode basically consists of a SA applied to an IP tunnel. Although this mode can also be applied to a host to host communication, it is especially useful when at least one of the ends of the security association is a security gateway. In fact, in that case the transport mode can not be employed.

Tunnel mode involves two IP headers: the outer one specifies the IPSec processing destination whereas the inner one identifies the ultimate (real) destination for the packet. This configuration is used in Virtual Private Networks (VPN).

3.9.2 The IP Authentication Header (AH)

IPSec uses two protocols to secure the information in transit: the IP Authentication Header (AH) and the Encapsulating Security Payload (ESP), which can be applied alone or in combination with each other to provide the set of security services required by the application. A SA can be used with one of these protocols, either AH or ESP, but not both at the same time. Whenever both AH and ESP protection needs to be applied to the same stream, al least two SAs must be established.

The AH protocol, specified in [25], provides connectionless integrity and data origin authentication for IP datagrams, as well as an optional anti-replay service. It can be used in either transport or tunnel mode, but the former is only available in host implementations. Protection is achieved by means of the AH header, that contains an Integrity Check Value (ICV) field. This field is a value computed over the IP datagram that can be used by the recipient to authenticate the sender and to check whether the information was modified in transit or not. It can be computed using either symmetric or asymmetric cryptographic algorithms.

In order to verify an inbound packet, the receiver computes the ICV over

the appropriate fields of the datagram, filling the mutable IP header field with zeros and using the specified authentication algorithm. If this value coincides with the ICV attached to the packet, then the datagram truly belongs to the purported sender.

In transport mode, the protection covers the entire payload (upper layer protocol data) and all those fields of the IP header which are not modified in transit[5]. Tunnel mode, however, encapsulates the IP datagram with a new IP header, so the protection extends to the payload and the whole header.

3.9.3 The IP Encapsulating Security Payload (ESP)

The ESP protocol [26] basically offers the same security services as AH, but it also provides a confidentiality option to allow data encryption. Thus, the main difference between ESP and AH is this additional service, as well as the coverage of the protection, as ESP does not protect any IP header field when used in transport mode. In tunnel mode, however, the whole IP datagram is covered.

The operation of the ESP protocol is very similar to AH for data origin authentication, connectionless integrity and anti-replay. Confidentiality is achieved by encrypting the payload and inserting the resulting ciphertext in the datagram instead of the plaintext. Although there are different mechanisms to encrypt the payload, ESP was specifically designed to work with symmetric algorithms, so it is highly recommended to use them. Figure 3.4 illustrates the structure of the AH and ESP headers.

3.9.4 Security databases

The protection afforded by IPSec to IP traffic is based on a set of requirements defined in two databases: Security Policy Database (SPD) and Security Association Database (SAD). The former specifies general policies and is applied to all IP traffic inbound or outbound from a host or security gateway. The latter is more specific and contains parameters associated with an individual SA.

The SPD indicates what security services are offered to IP datagrams and the way in which they are applied. It must be consulted for all inbound and outbound traffic, including non-IPSec traffic, and its function is to discriminate among the packets that need IPSec processing and those that are allowed to bypass it. There are actually three different processing choices for every inbound or outbound datagram: discard the packet, process it

[5]All those "mutable" fields that may be modified in transit can not be included, since their values are unpredictable for the recipient. In the IPv4 header these fields are Type of Service (ToS), Flags, Fragment Offset, Time to Live (TTL) and Header Checksum, and must be filled with zeros prior to the ICV computation.

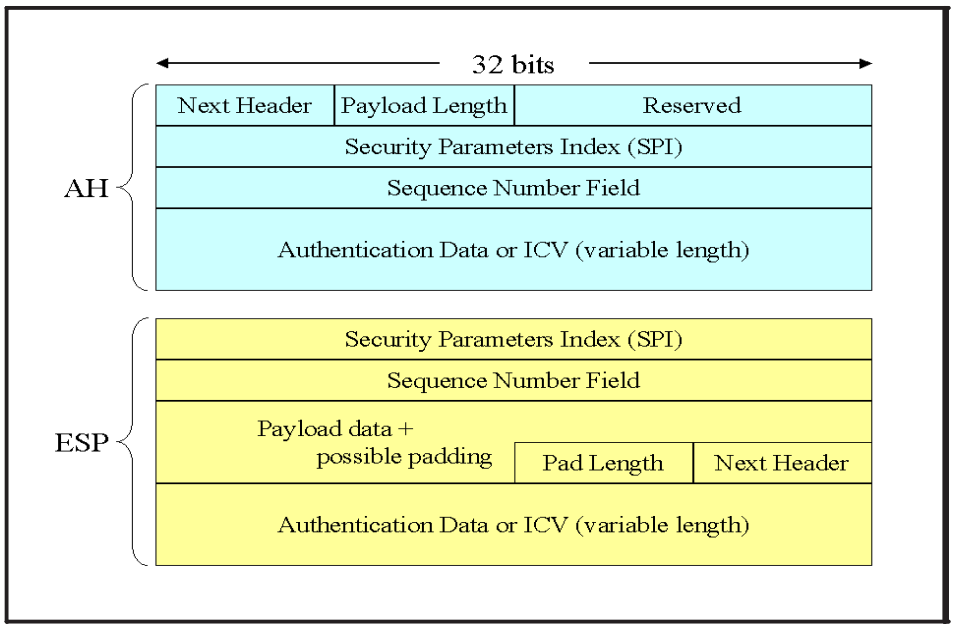

Figure 3.4: AS and ESP headers

applying IPSec or bypass. Whenever an IP packet needs to be afforded protection, the SPD specifies the services that should be provided as well as the protocols and algorithms to be used.

The SAD database defines the parameters associated to every singular SA. Therefore, each SA must have its own entry in the table.

3.9.5 Key distribution in IPSec

IPSec authentication, integrity and confidentiality services are usually based on symmetric cryptography, so they rely on secret shared keys. This implies that the IPSec architecture must be supported by some parallel key distribution mechanisms.

In order to address this requirement, the IPSec standard considers two different approaches that may be chosen depending on the features and constrains of the system:

- Manual key distribution. An administrator manually configures all the devices involved in the IPSec system and provides them with the relevant SA management data and with the secret keys. The main drawback of this approach is the lack of scalability, but it could be useful in small static systems.

- Automated SA and Key management. For larger systems there is a need for a mechanism to automatically distribute keys and to accom-

modate on-demand creation of SAs. The default mechanism to be used with IPSec, based on a public-key approach, is the IKE protocol, which is defined in [24].

3.10 HMAC algorithms

MAC (Message Authentication Codes) mechanisms make use of a secret shared key to provide integrity and data origin authentication of information that needs to be transmitted over unreliable channels, providing the secret key is only known by the communicating entities. Thus, they can be classified as symmetric algorithms, since the secret key must be known by both the sender and the receiver.

A Hash Message Authentication Code (HMAC) [27] is a specific MAC algorithm that uses two components to produce a digital signature of the message: a cryptographic hash function and a secret key. Hash functions themselves are not suitable to be used for authentication purposes, since they do not share any secret key between sender and receiver. HMAC algorithms combine the use of a hash function with a secret shared key, so that they can be used for authenticating the ends of the communication.

Any hash function is suitable to be used providing it performs well in software, but the most common are MD5 (HMAC-MD5) and SHA (HMAC-SHA). The latter is considered to be more secure, although it produces an output (160 bits) bigger than MD5 (128 bits).

Both HMAC-MD5 and HMAC-SHA-1 can be used as authentication mechanisms within the IPSec architecture, as explained in [28] and [29] respectively.

Chapter 4

SCSP performance improvements

In this chapter we discuss some performance improvements we have developed in order to speed up the overall operation of the SCSP protocol. The limitations of buffers implemented as linked lists are analyzed, and a new approach based on dbm databases is proposed. In addition, we address the message loss problem detected in the previous version.

4.1 Problem description

The proposed improvements are mainly focused on the Cache Alignment (CA) phase, which is by far the most critical in terms of execution time and CPU consumption. During this phase, two directly connected SCSP speakers exchange their database contents in order to synchronize them, so that they can provide clients with the same set of database records.

Since both servers are likely to have a considerable amount of initial entries, the replication process may take some time due to the heavy message exchange required. Furthermore, the alignment process involves a substantial number of database queries which can easily lead to a significant slowness in the protocol operation if they are not implemented following an efficient algorithm.

The performance results taken from [30] show that the alignment and replication times for the current SCSP implementation grow quadratically with the increase in the number of initial records. Alignment time can be defined as the time required for a peer to receive all the previously requested CSA records. However, the real replication process requires that all the reply messages (acknowledges) have been successfully received and processed. This time, longer by definition than the alignment time, is called replication time.

Figure 4.1 shows this behavior for different database sizes. The test environment in which these measures were taken is described in table 4.1.

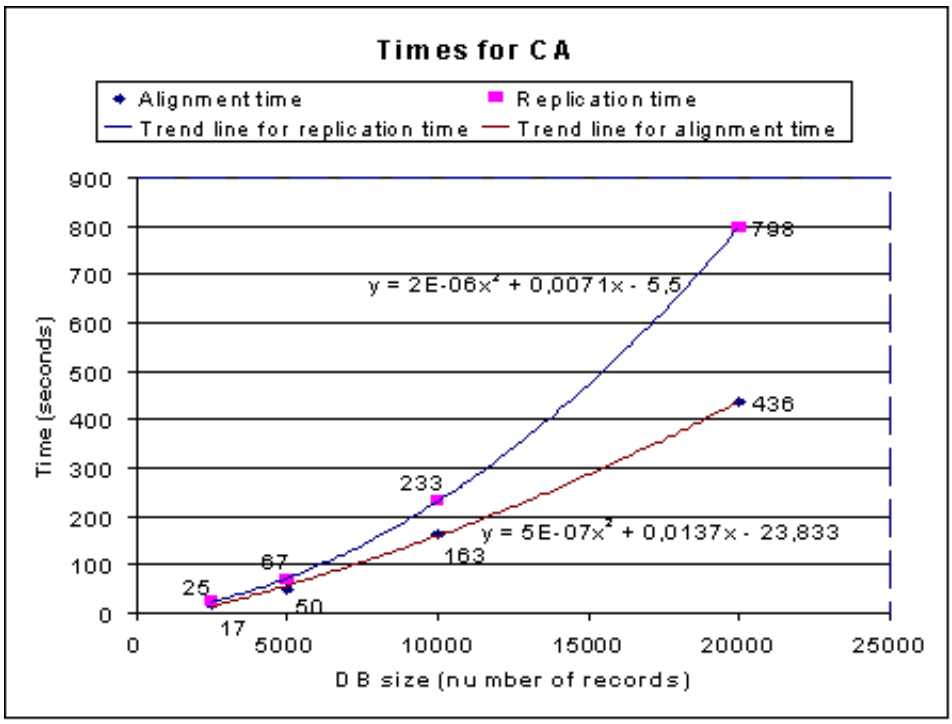

Figure 4.1: Times required for cache alignment (SCSP former version)

CSAS and CSA buffer implementation	**Linked-lists**
Cache key size	**9 bytes**
CSA size	**100 bytes**
Maximum number of entries	**20000**
Maximum SCSP packet size	**1000 bytes**
Number of servers	**2**
Topology	**Inline**

Table 4.1: Database size test description (SCSP former version)

These replication times are too high to be used in real environments. The quadratic trend line implies that it is unfeasible to work with more than 20.000 records per database, as the alignment and replication times would be unacceptable for any application. Thus, we can conclude that some performance improvements are needed in order to optimize the protocol implementation.

The main goal of this section is to identify those bottlenecks that cause the protocol to perform slowly and find suitable countermeasures to address

them. Notice that these problems were already detected in [30], so a brief description can also be found there.

We can summarize the implementation weaknesses as follows:

- During the Cache Alignment phase, two data structures are mainly used to perform the data exchange between two directly connected peers. Cache State Advertisement Summary (CSAS) records contain a summary of a database entry, and they include enough information for SCSP to decide whether one record is already present in the local cache and to compare the newness of the summary against the newness of the cached entry. CSAS are exchanged between peers at the beginning of the alignment process to let a server know about the contents of its peer's database. The server processes all the incoming CSAS and compares them with those stored in its local database, so that it can later request the missing or obsolete ones from the peer.

  Cache State Advertisement (CSA) records contain database entries, so they are used to exchange the information which is being replicated.

  Both CSAS and CSA records must be kept by SCSP in internal data structures. Since the protocol is coded in C language and these internal structures must be resized dynamically depending on the number of received and processed records (at execution time), they are implemented as linked lists. However, the seek algorithm of this kind of data structures has proven to be inefficient, since we have to go through the whole linked list comparing records until we find the specified one. Moreover, even if the record is not present in the structure, we have to inspect the whole list to realize it. The average seek time is approximately half the time required to go through the whole structure, and this time is strongly dependent on the size of the linked list.

  The conclusion is that the average seek time grows with the number of records kept in the list, and this is clearly unacceptable because it directly affects the scalability of the protocol.

- The second limitation is related to communications. The SCSP is an application-level protocol which requires the services of a transport protocol to carry out the message exchange between peers. Although either TCP or UDP might be used, SCSP is widely considered as a heavy protocol, so the current implementation uses UDP because of its lightness against TCP.

  Under heavy-load conditions, there is a severe packet loss in the operation. This loss is mainly due to the transmission of packet bursts in some phases of the protocol, which cause the UDP sender's buffer to overload and leads to an eventual packet discard. Even though the SCSP specification includes its own mechanism to cope with packet

loss by means of retransmissions, the implementation should prevent this data loss in buffers when possible, since retransmissions cause the protocol to work considerably slower.

4.2 Proposed countermeasures

4.2.1 CSAS and CSA buffers re-implementation

We have developed a new API in order to solve the described inefficiencies in the SCSP buffers. This new API is based on dbm databases, which are available in most UNIX implementations. Dbm libraries provide a simple database management facility that allows programs to store simple key-value pairs into files. The library provides the programmer with functions for database creation and management, as well as record insertion, extraction and deletion.

Internally, dbm files are implemented as arrays indexed by strings, also called hash tables. This means that any data item can be randomly accessed in constant time, no matter the position of the record within the table nor the database size, so this solution suits well for our purposes. Since the conventional dbm library is considered to be obsolete, the new GNU library gdbm has been used instead.

Although the implementation internally keeps many different CSAS and CSA buffers to deal with the SCSP complexity, only those that have been identified as critical for the performance have been re-implemented. However, the new buffer API has been designed to provide a programmer interface as similar as possible to the old one, so that all the remaining linked-list based buffers could be easily changed by slightly modifying the code. Notice that this would hardly improve the protocol performance, since the size of these buffers is not significant.

Figure 4.2 shows a performance comparison between the two approaches using CSAS buffers. The average seek time for linked-list buffers has been measured considering the extraction time for a record located in the middle of the list.

These experimental measures confirm that gdbm databases are capable of accessing records in a random way, with the seek time being constant and independent from the database size. On the other hand, average seek time increases linearly with the total number of entries in the linked-list based approach.

4.2.2 Solution to the UDP buffer overload problem

To overcome the UDP buffer overload problem on the sender's side, we have identified and re-implemented all the functions where packets were being

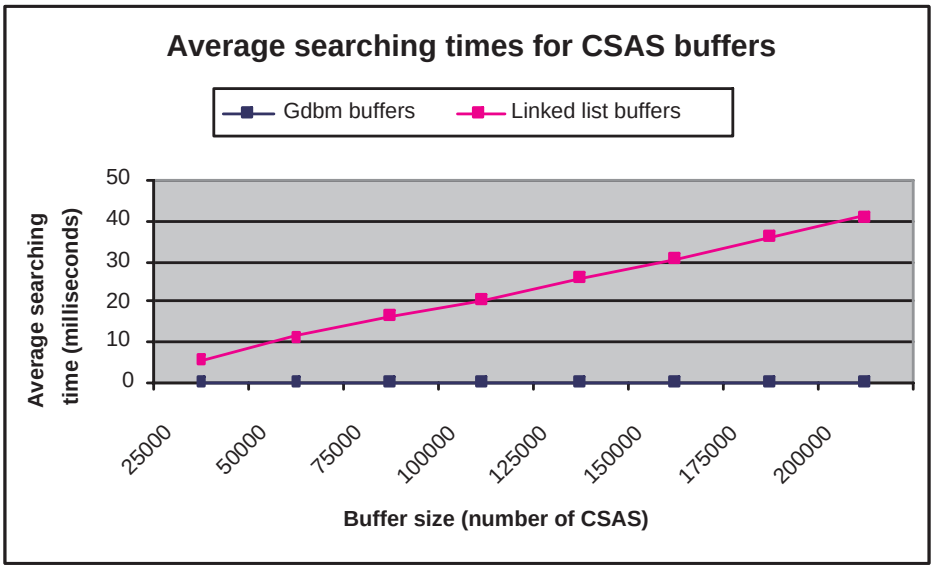

Figure 4.2: GDBM buffers vs linked-lists seek times

sent in bursts. The goal is to ensure that packets are gradually sent to the buffer in order to prevent buffer overload.

As an additional countermeasure, the internal UDP buffer size has been set to the maximum allowed by the operating system (250.000 bytes) on both sender and receiver sides.

4.3 Performance tests

Since the operation of the protocol is expected to improve with the implementation of the new countermeasures, we need to perform some tests to establish a performance comparison between the former SCSP version and the new one. All tests were performed using the UNIX machines WS17 and WS18 located in the lab. The main features of these machines are briefly described in table 4.2.

Machine name	WS17	WS18
Memory	256 M	256 M
Operating system	SunOS 5.6	SunOS 5.6
Number of processors	1	1
Processor type	UltraSPARC-II	UltraSPARC-II
Processor speed	300 MHz	300 MHz

Table 4.2: Description of the servers used for testing

4.3.1 Database size tests

The information about the new database size test environment can be found in table 4.3.

CSAS and CSA buffer implementation	**GNU gdbm library**
Cache key size	**9 bytes**
CSA size	**100 bytes**
Maximum number of entries	**100000**
Maximum SCSP packet size	**1000 bytes**
Number of servers	**2**
Topology	**Inline**

Table 4.3: Database size test description (SCSP enhanced version)

Results are depicted in figure 4.3, which shows that alignment and replication times have been considerably decreased in the enhanced SCSP implementation. Another important achievement lies on the fact that the response time to the growth in the number of initial entries has been linearized. This feature ensures that the protocol can perform well with large initial databases, and it is crucial for scalability.

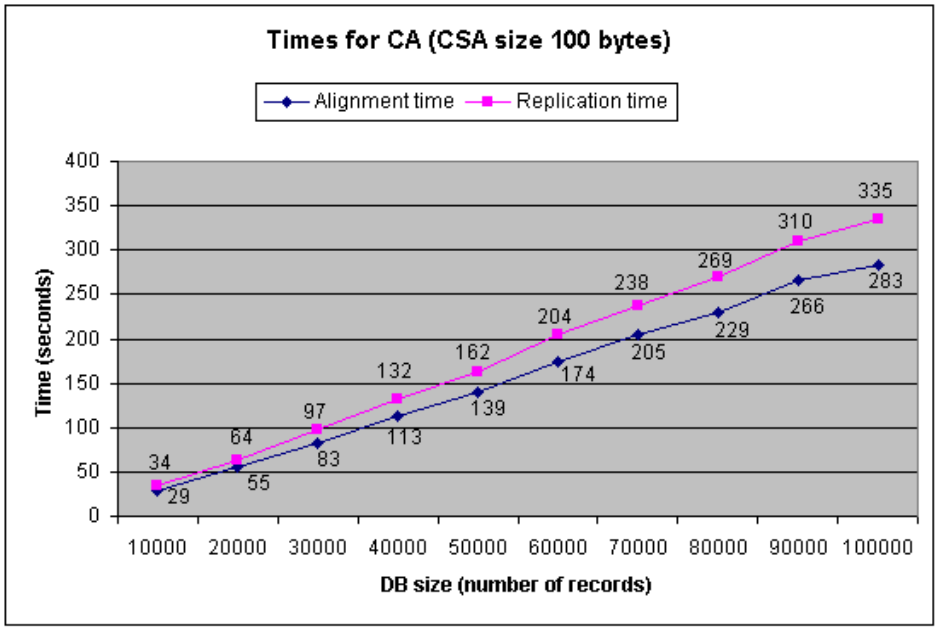

Figure 4.3: Times required for cache alignment (SCSP enhanced version)

In order to establish a performance comparison between both SCSP versions, we define a new parameter called improvement factor. This factor

is the quotient between the replication time of the former version and the replication time of the enhanced one. Figure 4.1 shows that the replication time in the former implementation when the size of the CSA is 100 bytes can be approximated by the following trend line:

$$Rep\_Time_{(Former)}(x) \approx 2 \times 10^{-6}x^2 + 7,1 \times 10^{-3}x - 5,5 \; seconds \quad (4.1)$$

where x is the number of records in the database. The equivalent expression for the enhanced version is:

$$Rep\_Time_{(Enhanced)}(x) \approx 3,4 \times 10^{-3}x \; seconds \quad (4.2)$$

Thus, the improvement factor can be calculated as:

$$\begin{aligned} Improvement\_Factor_{(CSA=100\ bytes)}(x) &= \frac{Rep\_Time_{(Former)}(x)}{Rep\_Time_{(Enhanced)}(x)} \\ &= \frac{2 \times 10^{-6}x^2 + 7,1 \times 10^{-3}x - 5,5}{3,4 \times 10^{-3}x} \end{aligned} \quad (4.3)$$

Since the former SCSP version does not have a linear response, the improvement factor depends on the database size (higher database sizes will result in higher improvement factors). Table 4.4 shows a comparison of replication times for both versions, considering different database sizes. We can see that the enhanced version is more than 60 times faster than the former one when databases of 100.000 records are used.

DB size (records)	**Replication time former SCSP version (seconds)**	**Replication time enhanced SCSP version (seconds)**	**Improvement factor**[a]
10.000	233	34	6,85
20.000	798	64	12,47
50.000	5.349	162	33,02
100.000	20.704	334	61,99

[a]CSA size=100 bytes.

Table 4.4: Replication time comparison

4.3.2 CSA data size tests

The previous test shows the protocol behavior for variable initial database sizes, but we must also verify that it performs properly for different CSA record sizes. The main variation in the protocol operation when we increase the CSA record size, providing that the maximum message size is fixed to

a value of 1.000 bytes, is that the replication process requires a heavier message exchange. The reason is that the number of CSA records that can be carried in a single message is smaller. Figure 4.4 shows the variation in the number of CSA records that can be allocated in a single message when the CSA record size is modified.

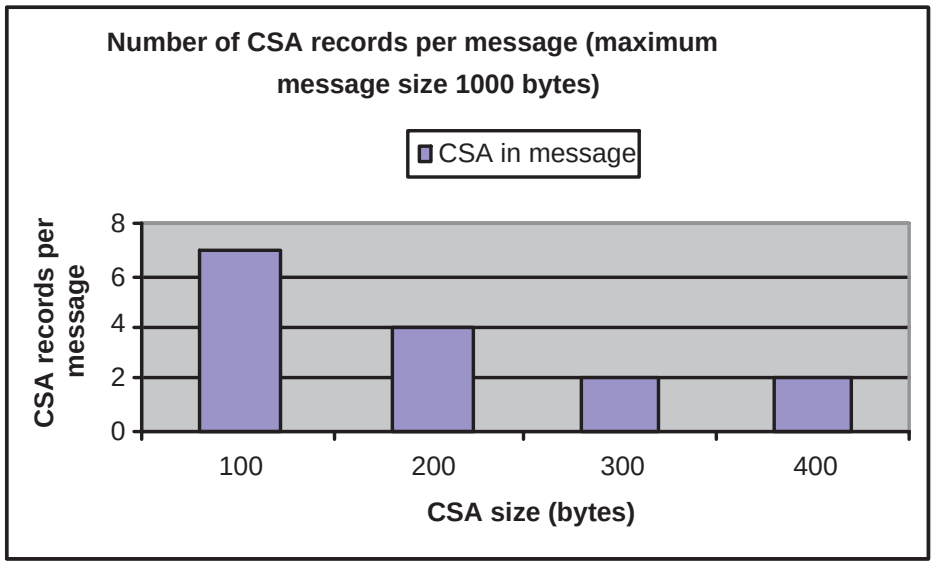

Figure 4.4: Number of CSA records per message

Alignment and replication times for different CSA sizes and for initial databases of 20.000 records are depicted in figure 4.5. Results show that there are two factors that influence the replication time:

- The number of messages to be processed.
- The record size. The number of messages exchanged for CSA sizes of 300 and 400 bytes is the same, since in both cases a single message can transport the same amount of CSA records. However, there is a 16% increase in the replication time. We conclude that this delay must be due to the increment in the CSA size (from 300 to 400 bytes).

4.3.3 CPU usage

Figure 4.6 depicts the CPU usage in a monoprocessor machine for the former SCSP implementation in the case of 20.000 initial entries (CSA size 100 bytes). These results have been taken from [30], and show that the percentage of CPU used is about 90% during the whole execution time. This is a consequence of the multi-thread approach followed in the implementation,

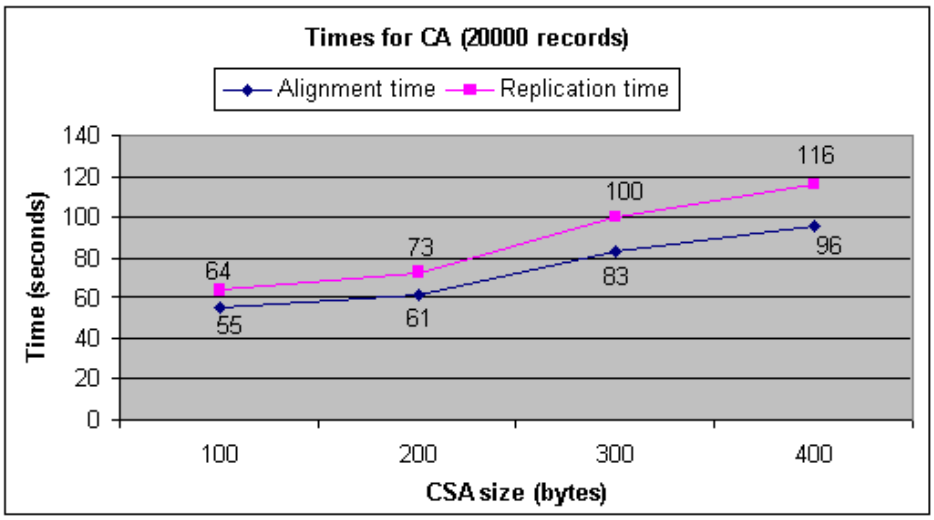

Figure 4.5: Times for cache alignment (CSA data size test)

and means that the high times needed to replicate the databases are due to slow processing and not to delays in I/O blocking operations.

The behavior of the gdbm-based version is depicted in figure 4.7 for initial databases of 100.000 records and CSA size of 100 bytes. The implementation of the internal buffers as gdbm files causes a remarkable increment in the number of I/O operations. This means that the percentage of processor usage can not be as high as in the former version. The machine spends a considerable amount of time waiting for I/O requests to the hard disk. This situation could be improved by using high-performance storage systems.

The first peak in figure 4.7 is caused by the initial entries load process. These entries are injected into SCSP through the external interface, which is implemented as a POSIX message queue. One thread is responsible for receiving all the records and inserting them into the corresponding internal structures. After that, the protocol enters the Cache Summarize phase, where peers exchange summaries of their database contents. During this phase, the pair of servers involved in the communication have at most one outstanding message, so the CPU load is very low. The situation changes during the Update State Cache, where the heaviest part of the message exchange takes place. To end up, we must remark that no packet loss was detected during the testing process.

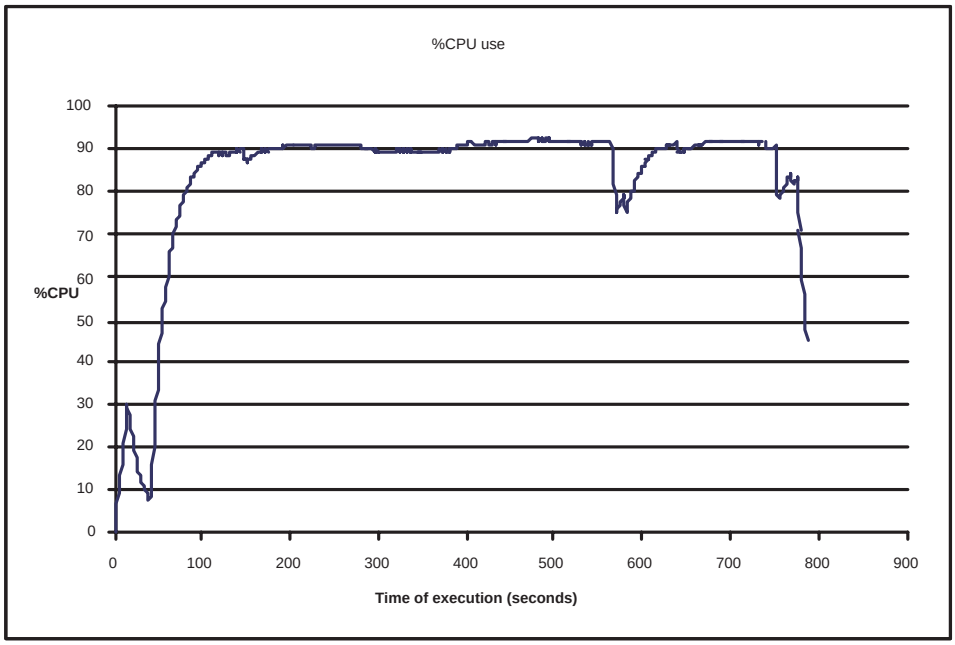

Figure 4.6: CPU consumption during cache alignment (SCSP former version)

4.4 Application of the results to the TRIP/SCSP system

We can use the performance results obtained in the previous section to make an estimation of the time needed to replicate the server databases in the TRIP/SCSP architecture. Following the assumptions and the number space analysis[1] made in [31], our system must be capable of replicating 840.000 records with 342 octets of CSA size in a reasonable time. The replication time for 20.000 records of the same length can be estimated from figure 4.5, by computing the average of the times needed to replicate the same amount of records with CSA sizes 300 and 400 bytes:

$$\begin{aligned} Rep\_Time_{(20.000)}(342) &\approx \frac{Rep\_Time_{(20.000)}(300) + Rep\_Time_{(20.000)}(400)}{2} \\ &= \frac{100\ seconds + 116\ seconds}{2} \\ &= 108\ seconds \end{aligned}$$

Since the enhanced SCSP implementation has a linear response, the time

[1]Number Portability region of 30.000.000 subscribers and directory numbers of 10 digits.

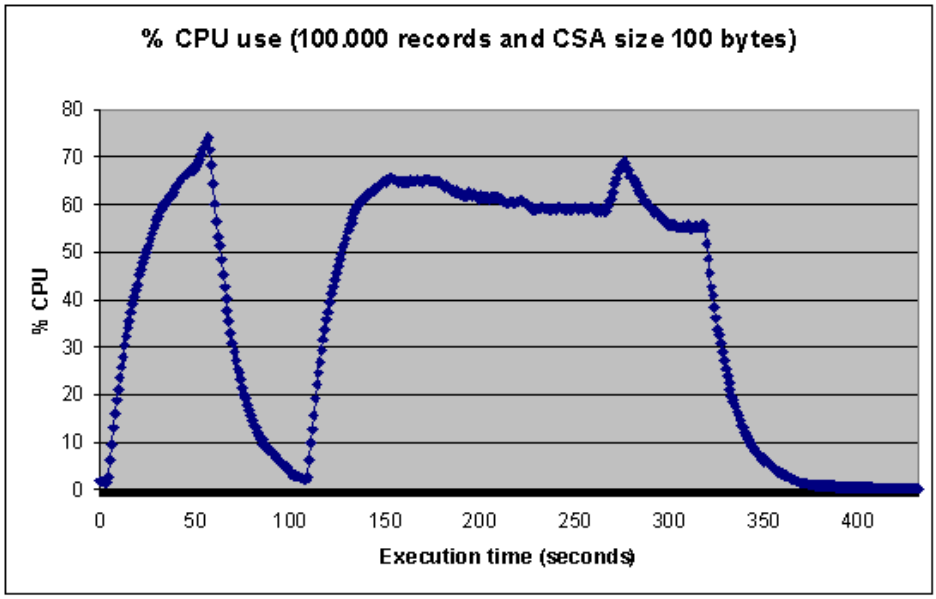

Figure 4.7: CPU consumption during cache alignment (SCSP enhanced version)

needed to replicate 840.000 records can be calculated as:

$$\begin{aligned} Rep\_Time_{(840.000)}(342) &= \frac{840.000}{20.000} \times Rep\_Time_{(20.000)}(342) \\ &= 42 \times 108 \; seconds \\ &= 4.536 \; seconds \\ &\approx 75 \; minutes \end{aligned}$$

Despite the improvements described in this chapter, the replication time is still too high to be used in a real environment. In order to obtain a usable system, this time should be reduced at least by a factor of 10. This would allow us to achieve replication times shorter than 10 minutes.

Chapter 5

Security countermeasures

This chapter describes the development of three security countermeasures aimed to secure the TRIP/SCSP architecture. The SCSP security module solves the TCP/IP related security problem and the TRIP speaker authentication, although an alternative solution based on IPSec is also presented. The solution for the TRIP level security problem is completed by both the PKI for TRIP speaker authorization and the TRIP authentication attribute.

5.1 SCSP security module

5.1.1 Design goals

TRIP speaker authentication is a necessary requirement, since every server must verify the identity of its direct peers before starting the data exchange. Even in a static configuration environment, where servers get the IP addresses of their peers from a configuration file, further verifications have to be carried out in order to authenticate peers, because the presence of an IP address in a file does not imply that the owner of that address has been authorized by a domain to exchange routing information with other domains. Furthermore, servers have to make sure that nobody is spoofing the IP addresses of their peers.

TCP/IP security is, however, a more complicated issue, since it usually involves different security services that can be used to address separate security requirements. Designers must decide the set of services that are needed by a specific application and the way in which they have to be applied in order to achieve the desired level of protection. Furthermore, the use of cryptographic algorithms is especially demanding in terms of computer power consumption, which implies that performance constrains could make it unfeasible to use some security services in many situations.

For instance, the SCSP protocol is used within our architecture to replicate the databases of remote TRIP peers. As we stated in chapter 4, the Cache Alignment phase takes a long time to complete when large databases are used, because replication requires a heavy message exchange. SCSP messages used during this phase have different functions depending on their type, and not all types carry vulnerable information. As an example, messages used to exchange CSAS records contain cache keys needed by peers to identify and index database entries. This information is useless for third-party attackers, but the slightest change in the message could corrupt one or more cache keys and make the recipient unable to process the message. Thus, this particular message type clearly requires an integrity service to prevent undesired modifications during transmission over the network, but the time needed to encrypt messages can be saved as confidentiality is not a primary concern.

This leads to the conclusion that security mechanisms regarding messages in transit must be implemented in a modular way, so that different message types can be afforded different security services depending on their requirements. These requirements determine the way in which every message type is processed, and are defined in a security policy that is checked whenever a message is sent to the peer or a new one is received from it.

5.1.2 Security services for SCSP traffic

We define three basic security services to protect the SCSP traffic: confidentiality, data integrity and data origin authentication. As we stated in the previous section, different message types may require different services. Thus, our design allows the security administrator to enable or disable them independently for every message type. This complies with the modular approach and provides the system with a great deal of flexibility when applying security policies.

Confidentiality may be selected independent of all other services, and is achieved by encrypting messages with a symmetric block cipher. We have chosen for this purpose the AES standard, based on the Rijndael cipher, as the default algorithm, but other ciphers can be easily added in future versions.

The AES algorithm is widely accepted as the current standard, and uses key lengths long enough to remain secure for many years. Although different key sizes are specified in the standard, our system uses 128-bit keys, which are considered to be unbreakable until the end of this century.

The only part of the SCSP messages that is not encrypted is the Fixed part, present in all messages and used by the recipient to determine the type and size of the received message. Since messages may have arbitrary sizes, AES is used in CFB mode to avoid padding.

Data origin authentication and integrity are joint services, which means

that they can not be selected independently. They are offered as an option to be used in conjunction with confidentiality, and their implementation is based on the SCSP authentication extension defined in [6]. This extension carries a HMAC signature computed over the entire message and requires the peers to share a secret symmetric key. The default algorithm supported by SCSP is HMAC-MD5, that produces 128-bit authentication values, but additional digests may be implemented if needed. In our case, we consider that MD5 is an obsolete and insecure hash function whose use should be avoided within our architecture. Therefore, we propose the redefinition of the extension and the adoption of the HMAC-SHA-1 algorithm, based on the SHA-1 hash function, which produces 160-bit outputs and is considered to be more secure and resistant to brute force attacks.

All SCSP extensions, as defined in [6], must comply with the format shown in figure 5.1. It consists of a triplet (Type, Length, Value), where Type specifies the extension type code (1 for the SCSP authentication extension). The Length field stores the length in octets of the Value field, which is different for each extension type. For the original SCSP authentication extension, this field stores a Security Parameter Index (SPI), used as an index into a key table, and the actual authentication data field (HMAC output). Since we do not need the SPI, our redefinition of the extension omits this field so that the Value field only stores the 160-bit HMAC-SHA-1 value. The redefined SCSP authentication extension is also illustrated in figure 5.1[1].

If one or more extensions are present in a message, the extension part is terminated by the End of Extensions extension, defined in [6].

Whenever a message requires this authentication/integrity service, the HMAC value is computed over the entire message, including the Fixed part, and appended to it as an authentication extension. If the message, however, requires both confidentiality and integrity/authentication, encryption is performed first, and it does not encompass the authentication extension. The HMAC authenticator is then computed over the whole message. This order allows for the possibility of parallel processing of packets at the receiver, because decryption can take place in parallel with authentication. Figure 5.2 shows the outbound packet processing when different combinations of security services are selected.

Since both confidentiality and message integrity/authentication services are based on symmetric algorithms (AES and HMAC-SHA-1), they require the exchange of secret keys between peers to operate properly. These services are applied on a peer to peer basis, which means that every single connection between two SCSP speakers uses its own private keys. Although the problem of secret key distribution over the Internet has been studied in depth during

[1] We assign the extension type 2 to the redefined extension.

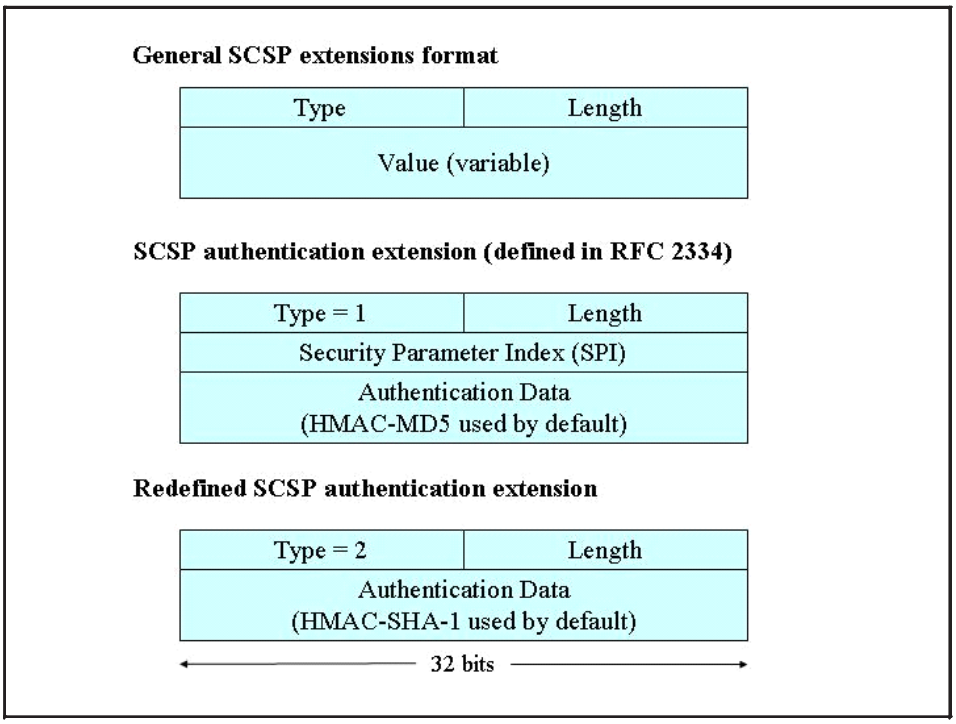

Figure 5.1: SCSP authentication extension redefinition

the last decade, and many key exchange protocols[2] have been proposed so far, our particular system can take advantage of the Hello phase to securely exchange the symmetric keys, as will be explained in the following section.

5.1.3 Hello Security extension

Hello messages are used to establish a bi-directional connection between two SCSP peers. Once the connection has been successfully set up, the nodes continue exchanging Hello messages as keep-alive notifications.

When a TRIP/SCSP server starts running, the IP addresses of its peers are stored in a configuration file. The first task that must be performed before starting the information exchange is to ensure that those peers are authorized TRIP speakers, certified by their respective ITADs to participate in the TRIP architecture and to export routes to other intra and inter domain peers. Since the Hello phase is used to set up the required connections between peers, it can be used to perform peer authentication as well. We may also use Hello messages to exchange the required secret keys between peers when either confidentiality or integrity/authentication have been selected to secure SCSP traffic.

[2]The most widespread key exchange protocol nowadays is IKE [24], used in IPSec.

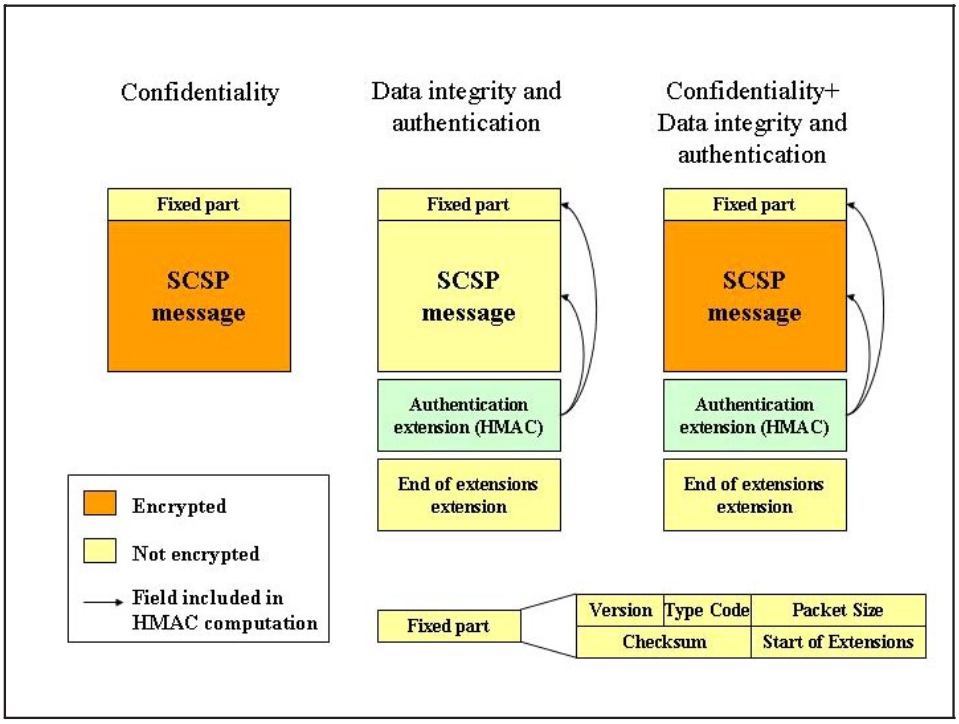

Figure 5.2: SCSP message processing for different security services selected

In order to achieve these two goals, we have defined a new SCSP Security extension to be included in every Hello message. The extension carries all the information needed to authenticate servers using a digital signature, and some additional fields concerning security services in SCSP traffic. These fields contain information about the algorithms used to implement the security services, and they also carry the required symmetric keys when needed.

The structure of the extension and the way in which it fits within the original Hello message format are depicted in figure 5.3.

The first 32 bits are the mandatory header that must be present in every SCSP extension. It consists of two fields:

- **Type.** The extension type code. The HMAC based authentication extension has been assigned the value 2 in the previous section, so we can use the value 3 for the Hello Security extension.
- **Length of the extension.** This value is variable since it depends on the size of the digital signature to be carried and the length of the (optional) encrypted keys.

The rest are security specific fields:

- **Hash Type.** Indicates the hash algorithm used to compress the message before being signed. The recipient needs this value to perform

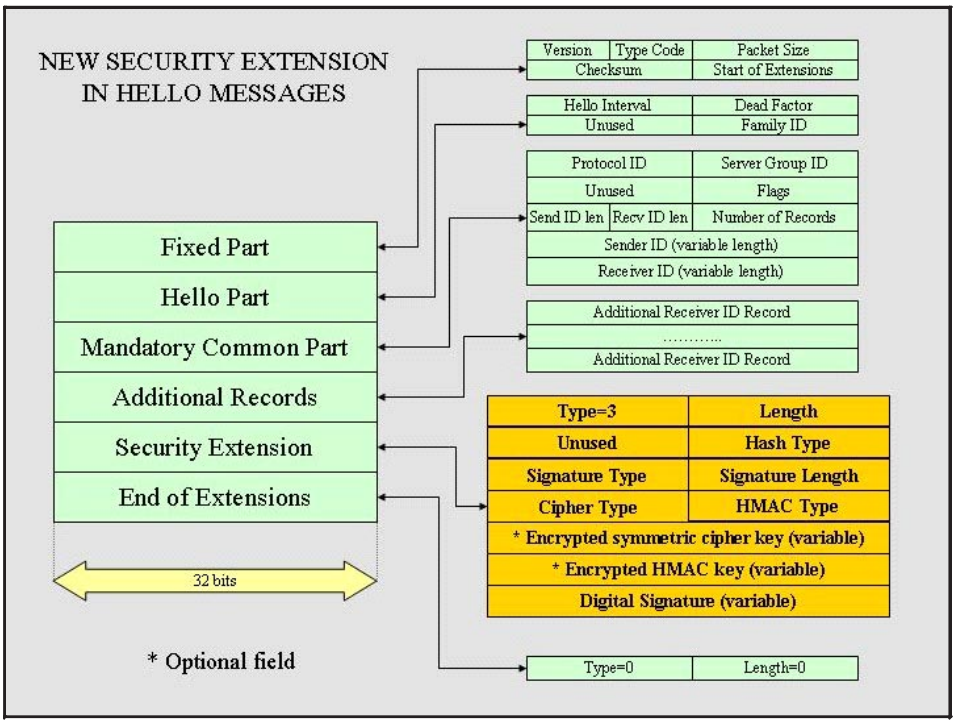

Figure 5.3: Hello message security extension

the verification process, as will be explained later. The codes already defined are:

  - MD2=1
  - MD5=2
  - SHA-1=3 (default)
  - RIPEMD=4

- **Signature Type.** Although the first version only implements RSA, other algorithms could be added in future versions. RSA has been selected as the default algorithm because it represents a de facto standard and it provides good security properties. The key length used is 1.149 bits and has been carefully chosen taking into account the RSA security study carried out in chapter 3. The Signature Type field allows the recipient to know which public key should be used to verify the message, in case the sender has different keys corresponding to different algorithms. The codes currently defined are:

  - RSA=1

- **Signature Length.** Total size in bytes of the digital signature carried in this packet. This value could vary depending on the algorithm being

used to sign the message and the key length. The recipient needs to know this value in order to unpack correctly the signature from the message.

- **Cipher Type.** Symmetric cipher used to encrypt messages in transit. As we explained before, the current version only supports AES-128, so the codes are:
  - Confidentiality service not selected=0
  - AES=1

  More ciphers can be easily added to future versions by defining new codes. However, AES is used now as the default algorithm.

- **HMAC Type.** Specifies the HMAC algorithm used for message integrity and data origin authentication. The current version considers three choices:
  - Integrity/Authentication service not selected=0
  - HMAC-MD5=1 (not recommended)
  - HMAC-SHA=2 (default)

- **Encrypted Symmetric Cipher Key.** Symmetric key used to encrypt and decrypt SCSP messages in transit. This field is present in the security extension only if the confidentiality service have been selected. Although the default algorithm, AES, can operate with different key lengths, the version used in our implementation works with 128-bit keys. Since the Hello message is sent to the peer without being encrypted, this field can not store the plain key. Thus, this key is first encrypted with the public key of the peer. This public key is extracted from the peer's X.509 digital certificate, which is obtained by using the PKI facilities described later in this chapter. By encrypting the symmetric key with the peer's public key we ensure that the peer is the only server able to get the decrypted version of the key, and thus the only one that will be able to decrypt the SCSP messages in transit. The size of the resulting ciphertext is the same than the signature length.

- **Encrypted HMAC Key.** Secret key needed to perform and verify HMAC authenticators. This field is present only if the integrity (and authentication) service has been selected, and its size is variable. As in the previous case, the key is first encrypted with the recipients's public key to hide it from intruders.

- **Digital Signature.** Stores the digital signature of the message. Its actual size depends on the length of the private key being used to sign

the information, but it usually varies between 128 and 256 bits for RSA signatures.

Whenever this Hello security extension is present in a SCSP Hello message, and in order to comply with the SCSP specification, it must be followed by the End of Extensions extension.

Outbound Hello packet processing

When a server needs to send a Hello message to one of its peers, it first fills the general SCSP headers and the Hello specific fields with the appropriate values. Then, the security extension is filled with the values corresponding to the security parameters that have been previously obtained from the security configuration file. This file is called "sec_conf" and provides the server with information about the signature algorithm and hash function to be used for peer authentication, as well as the algorithms regarding security services.

The presence in the security extension of the symmetric keys for both HMAC and encryption cipher is optional; they are needed only if the corresponding confidentiality and integrity/authentication services have been selected. A zero value in the HMAC Type or Cipher Type fields indicates that these features are not in use, and thus the recipient of the message must not expect to find the corresponding keys in the message.

When needed, these keys are generated randomly by the server and encrypted with the recipient's public key before being placed in the message. Key sizes are closely related to the algorithms in use. For instance, the proposed default symmetric cipher, AES-128, requires 128-bit keys. The key for HMAC can be of any length, although keys smaller than the length of the output are strongly discouraged since they would decrease the security strength of the function. Thus, for the proposed HMAC-SHA-1 algorithm, keys of 160 bits will be used.

Since the digital signature is not available until the end of the signature process, the Digital Signature field is first filled with zeros. Once the whole message has been built, including the End of Extensions extension, it is compressed using the selected hash algorithm and the resultant digest is then digitally signed with the private key of the server. Finally, the signature is placed on the security extension instead of the zeros.

By signing the entire message, we can ensure that any alteration caused by transmission errors or malicious attacks in any of the fields of the message will be detected by the recipient, so the message can be discarded. Therefore, although the main purpose of the digital signature carried in the Hello message is to allow peer authentication, it can be also used as an additional way to guarantee Hello message integrity.

Inbound Hello packet processing and key negotiation

Upon receipt of a Hello message, the recipient unpacks it to extract the different headers and fields. The Signature length field present in the Security extension allows it to extract the signature correctly. This signature is stored apart while its space in the message is filled with zeros, and the message digest is then computed over the entire packet using the hash function specified in the Hash Type field. The public key of the sender is extracted from the corresponding X.509v3 certificate, which has been previously obtained and validated using the PKI facilities, and used to verify the digital signature received with the message. The output of this decryption process is another digest that is checked against the computed one to ensure that the message truly belongs to the sender and that the message has not been modified during the transmission.

Once peer authentication has been carried out, and assuming that either confidentiality or integrity/authentication services have been selected to secure the SCSP session, the corresponding secret keys are extracted from the message and decrypted with the recipient's private key. Since every server has randomly generated its own symmetric keys, peers need a mechanism to agree about the ones that will be used during the session. This can be achieved by forcing the peer with a lower IP address[3] to discard its own keys and to adopt the ones received in the Hello message. This negotiation scheme makes use of the Hello phase to address the symmetric key distribution problem and avoids the use of other protocols to perform this task.

Another possible approach would be the use of different keys for each direction of the communication. A server generates its own symmetric keys and uses them to process all the outgoing information. The remote server receives these keys and uses them to process all the traffic received from the sender. This mechanism has two advantages over the previous one:

- If the security of one key is broken, only one direction of the communication is compromised.
- No key negotiation is needed, since every server makes use of both their own keys and the keys generated by their peers.

5.2 Public key infrastructure for TRIP server authorization

Our security architecture must be supported by a parallel mechanism to allow server authorization. Every TRIP server exchanging routing information with other servers must have been previously authorized to represent

[3]IP addresses are first converted into integers in order to perform the comparison.

its ITAD and to participate in the information exchange. This authorization procedure is carried out by means of digital certificates. TRIP speakers have to obtain these authorizations from their ITADs before setting up the corresponding connections with remote peers, so that authentication can take place during the Hello phase. Servers are also responsible for retrieving from a certificate server all the certificates required to authenticate their peers. Furthermore, TRIP speakers may also want to verify the identity of ITAD owners to ensure that their peers have been authorized by reliable organizations, and that these organizations have been assigned a valid ITAD number to operate within the IP telephony architecture.

In order to support this hierarchical authorization system, the security module makes use of a PKI based on X.509v3 certificates. The number of levels needed by this PKI can be easily determined by examining the current situation of the TRIP architecture. In this early stage, the Internet Assigned Numbers Authority (IANA) directly assigns ITAD number to organizations. These numbers may vary within the 0-65.535 range, although the sub-range 1-255 is reserved for private use. Requests for numbers must include information about the organization that administers the ITAD as well as some contact data, and must be directly submitted to the IANA.

This means that the PKI for TRIP server authorization must contain 3 levels and 3 different certificate types. The PKI structure is depicted in figure 5.4. The IANA plays the role of root CA, and is in charge of issuing ITAD numbers to organizations. These organizations are ITAD number owners, and they represent the first CA level in the hierarchy. Every ITAD owner is responsible for designating their own TRIP servers, identified by IP addresses or DNS names, to operate within the domain. Thus, the second and last hierarchy level is composed by servers that have been authorized by ITAD owners to participate in the routing information exchange.

The certificate types required by this PKI are described in table 5.1. They are based on the X.509v3 standard and make use of certificate extensions to bind some application specific data about the subjects.

To authenticate a peer, a TRIP server must verify the entire certification path. If we assume that the verifier permanently stores a copy of the IANA's public key, this process implies the retrieval of two certificates from the certificate server: the TRIP server certificate of the peer and the one belonging to the owner of the peer's ITAD. Therefore, two signature verifications must be performed to authenticate every peer.

5.2.1 Certificate distribution

TRIP speakers are responsible for obtaining all the required certificates during the initialization process to ensure that all peers can be authenticated in the Hello phase. There are many ways in which a node can get the certificates. In the simplest scheme, the server stores local copies of all the

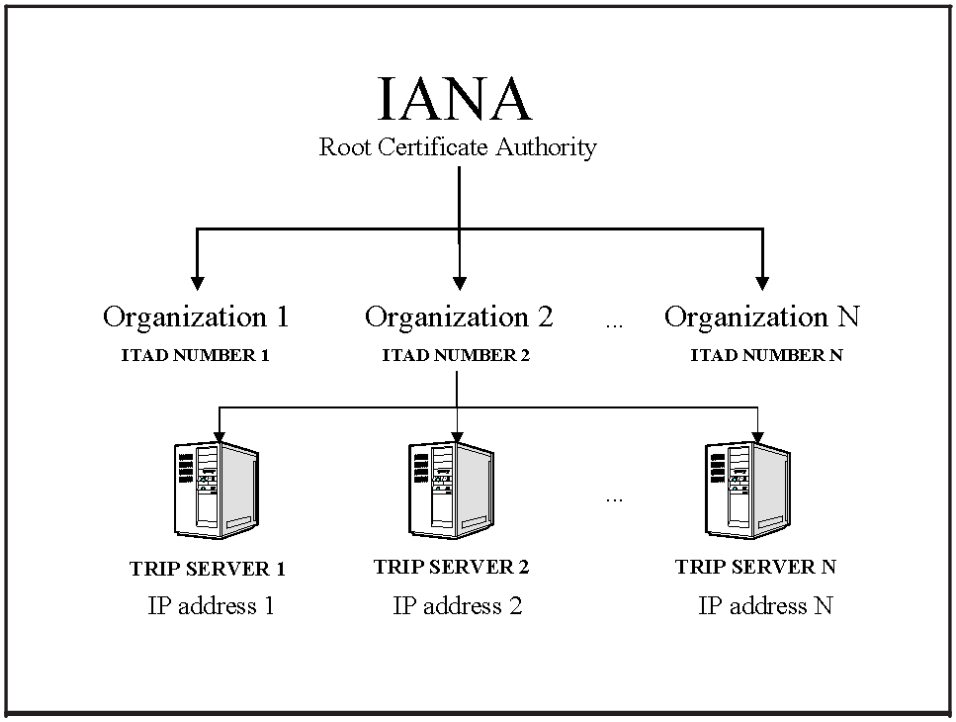

Figure 5.4: PKI for TRIP server authorization

existing certificates, including both TRIP server and ITAD owner certificates. The main disadvantage of this approach lies on the space needed to store the certificate database. The number of organizations owning ITAD numbers is likely to grow quickly in the future, as well as the amount of TRIP servers taking part in the architecture. If the number of certificates grows excessively, it can be unfeasible to maintain the whole database in disk or memory. Moreover, additional space is needed to store the CRL, that must be checked to ensure that certificates have not been revoked, and this CRL would need to be updated periodically from a central server.

The best way to address the certificate distribution problem is by means of a certificate repository. Both the certificate database and the CRL are stored in the repository, and TRIP servers can send requests to the server whenever they need to retrieve a certificate. This approach improves system scalability and allows for a centralized management of the CRL, ensuring consistency of the information.

PKI repositories are usually implemented using either DNS servers with security extensions, as described in [32], or LDAP servers [33].

Certificate Type	**Root**
Issuer	IANA
Subject	IANA
Description	Self-signed certificate that certifies the IANA as the root of the certificate chain.
X.509v3 extensions	No
Certificate Type	**Organization**
Issuer	IANA
Subject	Organization (ITAD owner)
Description	Certificate issued by the IANA to an organization that owns an ITAD number. This certificate type is not primarily used to verify the identity of the subject, but to prove ownership of the domain number.
X.509v3 extensions	One extension specifying the ITAD number assigned to the organization.
Certificate Type	**TRIP server**
Issuer	Organization (ITAD owner)
Subject	TRIP server
Description	Used by an organization that owns an ITAD number to designate a representative TRIP speaker.
X.509v3 extensions	One extension containing the IP address of the subject TRIP server (DNS could be used instead) and the ITAD number.

Table 5.1: Certificate types required by the PKI

5.2.2 X.509v3 certificate extensions

This section defines the X.509v3 extensions required by the PKI and described in table 5.1. Extensions must always include two components: an OID and a ASN.1 syntax structure. The basic structure is shown below:

```
Extension ::=
SEQUENCE  {
    extnID      OBJECT IDENTIFIER  ,
    critical    BOOLEAN DEFAULT FALSE  ,
    extnValue   OCTET STRING  }
```

The OID is stored in the extnID field, and the ASN.1 encoded structure is the value of the octet string extnValue. The boolean critical must also appear in every extension, and it is set to false by default.

Extension for TRIP server type certificates

The extnValue field is:

```
TRIPServerExtension ::= SEQUENCE  {
    ITADNumber  OCTET STRING,       # For example 3245
    ServerID    OCTET STRING  }     # For example 123.12.34.56
```

Extension for Organization type certificates

The extnValue field is:

```
ITADOwnerExtension ::= SEQUENCE  {
    ITADNumber1   OCTET STRING,      # For example 3245
    ...
    ITADNumberN   OCTET STRING  }
```

Notice that this arrangement allows the presence of several ITAD numbers in the certificate. This feature is useful when one organization has been allocated more than one ITAD number.

5.3 TRIP Authentication Attribute

Our security architecture uses the TRIP Authentication attribute, described in [34], to address the problem of data integrity over different TRIP hops. This mechanism allows the recipient of a TRIP message to verify the originator of an attribute and to make sure that it has not been modified by an intermediate LS.

Attribute authentication is performed by including a digital signature of the specified attribute. Therefore, the Authentication attribute includes as many signatures as TRIP attributes have been signed by the originator.

The specification of the Authentication attribute in [34] considers both RSA and DSA to be used as digital signature algorithms. Although every real LS must support both in order to ensure iteroperability, our prototype only supports RSA, since it will be mainly used for testing.

This third security countermeasure solves the TRIP layer related security (in addition to the Hello Security extension, which addresses the TRIP speaker authentication problem by carrying a digital signature, and the PKI for server authorization), and thus completes the security of the overall architecture presented in this document.

5.4 Performance tests with the security module

Encryption and decryption, as well as HMAC computations, are demanding tasks in terms of CPU consumption, so SCSP performance is expected to

worsen when messages in transit are being afforded security. This protocol slowness due to security services is proportional to the number of packets being secured as well as the set of services applied to each message type.

The highest security level within our design can be achieved when all message types are afforded both confidentiality and integrity/authentication. However, this ideal configuration would dramatically slow down the protocol operation, since the time needed by servers to process messages would be unacceptably high. Moreover, as we stated earlier, not all message types require the same set of security services, since the nature of the carried information varies from one type to another. Table 5.2 shows the message types currently defined in SCSP and a brief description about their functions and contents.

Message Type	**Code**	**Description**
CA	1	Used by peers to synchronize their entire initial databases. Basically contains CSAS records.
CSU Request	2	Aimed to update the state of database entries in servers that are directly connected to the node generating the update. It is sent to the peers when the server becomes aware of a change in state of a entry. Contains CSA records.
CSU Reply	3	Sent from a server to its peer to acknowledge one or more CSA records that were received in a previous CSU Request. Transports CSAS records.
CSUS	4	Allows one server to solicit CSA records not present in the local database but owned by the peer. Primarily carries CSAS records.
Hello	5	Its goal is to establish bi-directional connections between directly connected peers and to monitor the state of communications.

Table 5.2: SCSP message types description

The security services provided by our security module encompass message types 1-4. Types 1, 3 and 4 do not carry any application layer information, and are used to transport entry summaries (CSAS) between peers and to index these entries in the database. On the other hand, message type 2 transports upper layer information encapsulated in CSA records.

An adequate security policy must achieve a balance between security and performance. This means that, although all message types have to be afforded proper security services according to their requirements, the

protocol must keep working properly and reasonably fast.

5.4.1 Tested scenarios

In order to find a suitable security policy for the architecture, we have tested different security configurations in terms of replication time and CPU performance. Tests have focused on the SCSP Cache Alignment phase, the most critical in terms of number of messages generated and processed.

The symmetric cipher used in the tests is AES with the key length of 128 bits, whereas the algorithm chosen for the authentication extension is HMAC-SHA-1. Finding good implementations of these algorithms was not an easy task. We did some research about the current cryptographic libraries available for the C language. Although there are many of them in the public domain, they usually have a rather low performance in terms of speed. We have finally chosen OpenSSL and Cryptlib, which perform reasonably fast. We have carried out some speed tests for these libraries. The results can be found in Appendix A.

We have defined and tested four basic scenarios to understand how the security countermeasures proposed in this chapter affect the behavior of the protocol.

- **Scenario 1.** This scenario does not define any security policy, since messages are sent to the network without being protected by any security service. It is basically used to serve as a reference to other scenarios.

- **Scenario 2.** All messages in transit are afforded the authentication/integrity service provided by the HMAC based authentication extension. Confidentiality is not applied.

- **Scenario 3.** All messages in transit are afforded the authentication/integrity service, but only CSU Requests are encrypted. This configuration makes sense because CSU Requests are the only messages that carry upper layer related information, so they require encryption to hide this sensitive data from attackers. In contrast, message types 1, 3 and 4 carry SCSP related information worthless for intruders, so they only need the services provided by the authentication extension.

- **Scenario 4.** All messages are afforded full security, including both confidentiality and authentication/integrity.

The four scenarios are summarized in table 5.3.

Table 5.4 shows the alignment and replication times needed to synchronize initial databases of 20.000 records with CSA size of 9 bytes for the four scenarios under consideration.

Message Type	**Scenario 1**		**Scenario 2**		**Scenario 3**		**Scenario 4**	
	I/A[a]	C[b]	I/A	C	I/A	C	I/A	C
Type 1			•		•		•	•
Type 2			•		•	•	•	•
Type 3			•		•		•	•
Type 4			•		•		•	•

[a]Integrity/Authentication
[b]Confidentiality

Table 5.3: Security scenarios for testing

Scenario	**Alignment time (seconds)**	**Replication time (seconds)**	**Overhead ratio[a]**
Scenario 1	56	64	1
Scenario 2	63	72	1,125
Scenario 3	65	73	1,140
Scenario 4	71	80	1,250

[a]Overhead ratio=Replication time scenario n/Replication time scenario 1

Table 5.4: Times for cache alignment for the four tested scenarios

Figure 5.5 represents the percentage of CPU consumption for all the scenarios.

Both table 5.4 and figure 5.5 show that the addition of the SCSP security services to all messages does not affect dramatically the overall functioning of the protocol, since the system overhead is only 25%. The overhead due to the authentication extension is very similar to the one caused by the encryption-decryption process.

5.5 IPSec based security module

The security module described in section 5.1 is a built-in solution to the security problem; the functions needed to implement the security services are integrated into the native SCSP code, and they afford security to the SCSP application layer.

In this section we briefly describe a different approach to implement the security module based on the IPSec protocol suite. Obviously, the goals of this approach are the same as those defined for the built-in solution: provide a mechanism to authenticate TRIP speakers and address the TCP/IP layer security problem by protecting the sensitive information sent to the network. The proposed system consists of two elements:

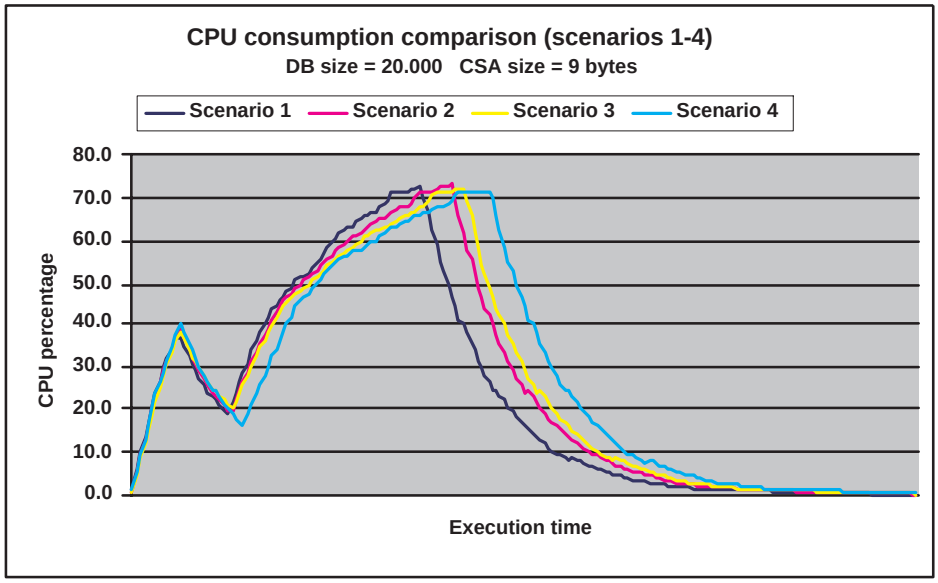

Figure 5.5: CPU performance comparison for the security scenarios

- As in the built-in solution, the PKI described in section 5.2 is used to address the TRIP speaker authorization problem. Before joining the TRIP architecture, every server must be authorized by its ITAD to participate in the information exchange and must obtain a signed certificate.

  Hello messages are used to carry the digital signature needed for peer authentication, but no secret keys are negotiated during the Hello phase. Thus, the optional fields reserved to transport the encrypted cipher and HMAC keys are not present, since confidentiality and message integrity/authentication are not provided by the SCSP layer, but by the IP layer. This means that in this approach the Hello phase is responsible for TRIP speaker authentication, but not for key negotiation. Upon receipt of a Hello message, and providing that the sender has not been authenticated before, a server retrieves from the PKI all the certificates needed and verifies the entire certification path for the peer.

- IPSec is used to secure the SCSP traffic. The selected protocol used to afford security to the SCSP messages is ESP, which mainly provides confidentiality, data origin authentication and integrity. AH is discarded because it does not provide confidentiality. The main difference between the authentication services offered by AH and ESP is the extent of the coverage. The former one protects both the payload and the immutable fields of the IP header, but the latter only

covers the payload. However, this drawback can be avoided by using the ESP protocol in tunnel mode, so that the entire inner IP header is also protected.

In order to secure the bi-directional traffic between peers, two host-to-host SAs must be established for every peer to peer relationship, one in each direction. We must notice that in this approach the keys required by the integrity/authentication and confidentiality services must be exchanged using a separate mechanism. IKE is the current standard protocol selected to perform this task within the IPSec environment, and can be additionally used to accommodate on-demand creation of SAs. This feature can be useful if an auto-configuration mechanism is implemented in the TRIP/SCSP architecture, since it could be used to secure new SCSP connections created dynamically.

The main advantage of the IPSec based approach is that the responsibility for protecting the information rests on the network layer, thus affording full protection to all SCSP traffic (including Hello messages) and to the UDP datagrams used to transport this traffic.

On the other hand, it lacks the granularity provided by the built-in approach. All traffic within the same SA is afforded the same protection level, and security services can not be applied independently depending on the application layer message type. Moreover, all servers taking part in the architecture must support an IPSec implementation.

Chapter 6

SCSP auto-configuration module

The main purpose of the SCSP auto-configuration module is to address the static configuration problem described in chapter 2 by providing the SCSP protocol with an efficient auto-configuration mechanism. Such a mechanism must be able to handle dynamic changes in both intra and inter domain topologies when needed, so that servers can be added to existing SGs or new SGs can be created on-demand without any manual intervention.

6.1 Design goals

Reliability is a primary concern when designing the module. Nodes that require auto-configuration services can not rely on the presence of central servers to obtain the topology information, since this would create single failure points within the architecture. Instead, the mechanism must be as decentralized as possible and all the required information should not be available on central databases, but distributed on other existing SCSP servers. This approach also improves the scalability of all those architectures that are using SCSP as the underlying replication service.

The second point to take into consideration is security. The SCSP security module described in chapter 5 is aimed to improve the security of the SCSP based architectures. Assuming that the module works properly within a statically configured environment, we must ensure that the introduction of automatic mechanisms to deal with auto-configuration does not result in negative repercussions for the system.

Transport layer independence is also an important issue. SCSP was designed to operate on Non Broadcast Multiple Access (NBMA) networks, so it is independent from the underlying transport protocol. Any module aimed to enhance the protocol must maintain this independence. However, the

auto-configuration module discussed in this chapter imposes an additional limitation to the transport layer; it must provide a multicast mechanism to carry out the Neighbor Discovery and Gateway Discovery phases, as will be explained later. Thus, the module is suitable to work with any transport mechanism that complies with this restriction. Since our existing SCSP implementation uses TCP/UDP to transport messages between servers, we will also present some optimizations for this particular case.

The SCSP protocol is always used by other applications or protocols that require database replication. Thus, we can consider it as a low level layer providing replication services to the upper or application layer. Primitive operations and services are offered to the application layer by means of the application interface between both levels. The interface defines the available services and the Service Access Points (SAP) through which they can be invoked.

In the architecture presented in this thesis, TRIP represents the application layer. However, we have tried to design an open and extensively configurable auto-configuration module that can be used by any SCSP based architecture, no matter the nature of the upper layer application.

6.2 Application interface extension

The main functionality of SCSP is database replication, so the most relevant primitives available in the interface are related to that function (update record or erase record, for instance). The auto-configuration module can be thought of as an additional set of services provided by SCSP to the application layer. These services are invoked using new primitives and they expand the functionality of the SCSP layer.

The current SCSP implementation communicates with the upper layer by means of a message queue based data interface, as described in [30]. It consists of three message queues that deal separately with local updates, remote updates and erases. This interface has been extended with two additional queues (one for each direction) to allow the bi-directional exchange of control messages between the SCSP layer and the application layer. This control interface expands the functionality of the existing data interface and can be used by the auto-configuration module and other future SCSP modules.

The structure of the extended interface is depicted in figure 6.1. Figure 6.2 details the six new interface messages needed by the auto-configuration module. The functionality of these messages will be explained throughout this chapter.

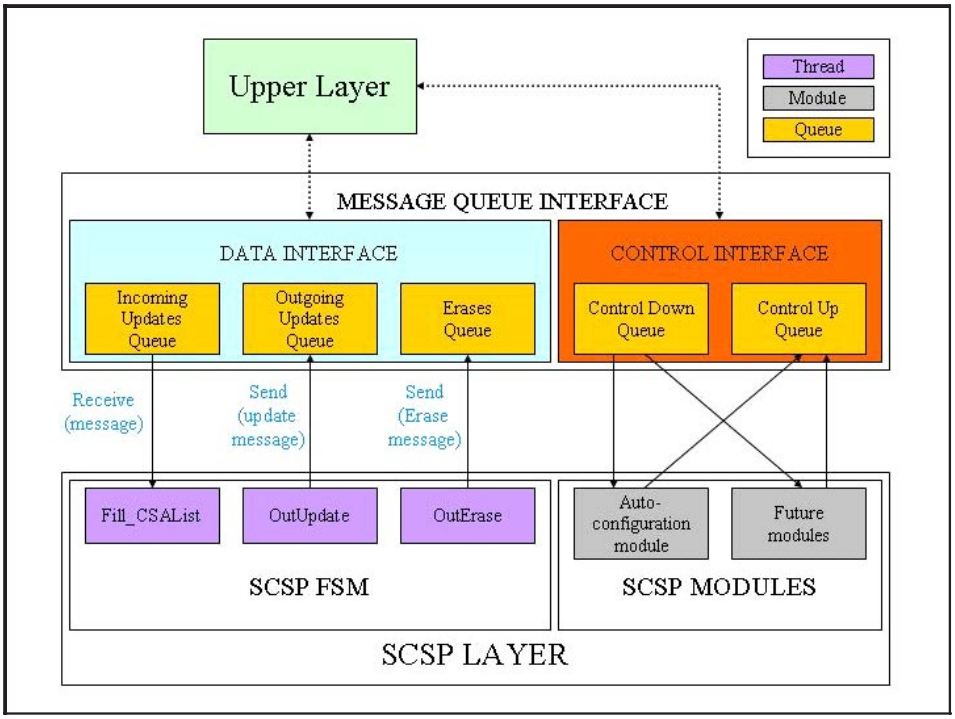

Figure 6.1: Application interface extension

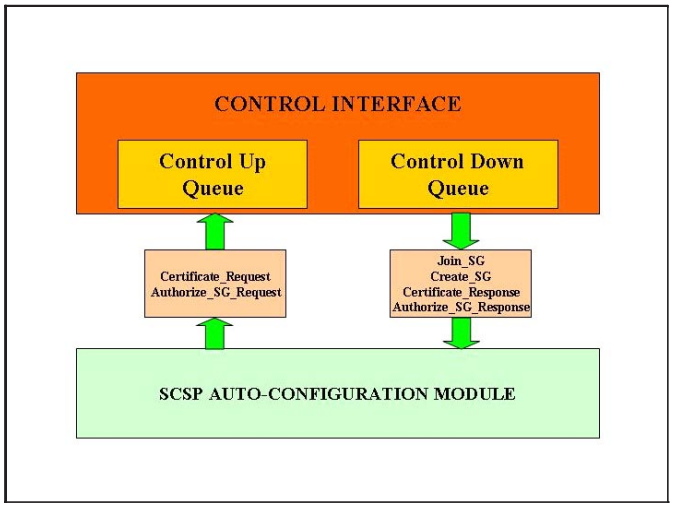

Figure 6.2: Primitives used by the auto-configuration module

6.3 Static configuration file

Every server obtains the configuration parameters from the "hosts" file, which consists of a set of columns with information about the server's DCSs. Nodes belonging to more than one SG have a set of peers for each group.

The configuration file begins with information about the port and the packet size of the system. The current version uses the same values for all the SGs, but it would be feasible to have different values depending on the group. Thus, a server replicating in two groups at the same time could be using different ports and packet sizes for each group.

Every column contains the address of the DCS, the SG, and some other SCSP related parameters. This arrangement allows for a customized parameter set for every single peer. The SCSP related parameters refer to timer values needed by the SCSP state machines, and their description can be found in [6]. A more detailed explanation of the "hosts" file format is given in both [30] and [35].

6.4 Auto-configuration services description

The proposed auto-configuration module comprises two basic services: intra-SG and inter-SG. The former one is aimed to facilitate the on-demand addition of servers to an existing SG, so its scope is limited to a single replication group. Since this is a very general mechanism designed to deal with the expansion of pre-existing SGs, it can be used by any SCSP based architecture.

The inter-SG mechanism, however, addresses a more specific problem closely related to the TRIP/SCSP architecture. As explained in chapter 1, every ITAD makes use of at least one internal SG to perform the intra-domain synchronization. All SCSP nodes taking part in the ITAD must belong to the intra-domain SG. An operator may decide to further divide its ITAD into areas. In this case, there is a separate SG to perform the synchronization within each area, and additional SGs, called inter-area SGs, are set up to distribute the routing information between two or more areas. Inter-domain agreements between different operators also use additional SGs, known as inter-domain SGs, whose members are representatives of the ITADs involved in the agreement.

The main goal of the inter-SG mechanism is to accommodate on-demand creation of inter-area and inter-domain relationships. This task is usually more complicated than the intra-SG auto-configuration, since it involves the dynamic establishment of SGs.

6.5 Intra-SG auto-configuration service

6.5.1 Server Group topology considerations

The purpose of the intra-domain auto-configuration mechanism is to accommodate on-demand addition of SCSP servers to pre-existing SGs. The new server must be able to select a set of DCSs to connect to within the SG. As we stated in chapter 2, SCSP places no topological requirements on SGs, which means that servers do not need to be connected in a fully-meshed fashion. The number of connection links in the group determines both the amount of traffic exchanged and the reliability of the communications. By minimizing the number of links we can reduce the amount of messages needed to propagate updates, but the reliability of the group is negatively affected. When servers are connected by many different paths, the disconnection of one node does not affect the operation of the rest of the group, since it is always possible to find alternative paths to route the information. On the other hand, weakly connected groups are more vulnerable to node failures since they can easily lead to the disconnection of the group.

The internal topology of a SG can be represented as an undirected graph, which is a simple diagram consisting of points (vertices) connected by lines (edges) [36]. In our case, the vertices represent SCSP servers and the edges are the direct connections between peers. In order to ensure that the information is replicated to all servers within a group, we must guarantee that the graph representing the group is connected, which means that there exists at least one path between every pair of vertices. The parameter that measures "how connected" is a graph is the connectivity, and gives an idea about how many elements should be removed from the graph to disconnect it. Within this model, server failure can be represented as the removal of the corresponding node from the graph.

A graph is said to be $k-connected$ if there does not exist a set of $k-1$ vertices whose removal disconnects the graph. Graph connectivity can be calculated using network flow techniques [36]. Thus, we can assume that the reliability of the SG directly depends on the connectivity of the graph that represents it. For instance, having a 2-connected graph ensures that the SG tolerates one node failure without being disconnected, and replication keeps working properly between the other servers in the group.

Group topology also affects replication performance and consistency, as proven in [30]. Both the number of elements in the group and the way in which they are connected determine the information flow within the group. Every topology can be considered as a composition of two basic models: star and inline. The evaluation tests carried out in [30] show that the performance of these models strongly depends on the update rate, i.e. the number of updates per second sent to the group. The star topology performs well for low update rates, since messages have to traverse fewer hops to reach their

destinations. The intermediate node (the one located in the middle of the star) is responsible for forwarding the updates generated in the neighboring servers. On the other hand, high update rates make this intermediate node to saturate quickly, worsening the replication performance considerably. In such cases it is more advisable not to focus the entire processing load on one single server. This can be achieved by using the inline topology, even though messages have to traverse more hops on average.

We can conclude that connectivity must be considered a fundamental parameter when designing SG topologies. It has to be carefully chosen to achieve a reasonable balance between traffic load and reliability. The new server runs the auto-configuration procedure to discover the other existing servers in the group and to decide the subset of servers it will connect to. Assuming that the existing group is k-connected, we must ensure that the addition of the new server does not modify the connectivity of the group (i.e. the group maintains at least the same connectivity after the node addition). This will normally be achieved by choosing an appropriate number of peers.

Furthermore, the design must take into account the nature of the upper layer application that is using the SCSP replication services, and characterize the update rates it will generate. This information is useful to decide whether to base our design on inline or star topologies.

6.5.2 Neighbor Discovery phase

The auto-configuration procedure begins with an unconfigured SCSP server with transport level connectivity. The "unconfigured" term means that it does not know any peer and does not have any configuration file. The procedure is triggered by the Join_SG signal, which is sent to the control interface by the application layer. This signal contains the SG identifier (SG_ID) of the SG the server has to be added to.

Upon receipt of the signal, the node enters the Neighbor Discovery phase. It starts sending Neighbor Discovery messages encapsulated in transport level multicast packets to the SCSP auto-configuration multicast address[1]. Thus, this phase requires that the NBMA network where SCSP is running provides multicast capabilities. Neighbor Discovery messages, which are identified by the SCSP type code 6[2], contain the SG_ID parameter. The already configured SCSP servers run a special thread that listens for incoming messages in the same multicast address. Although all of them are supposed to receive the Neighbor Discovery message, only those that belong to the SG_ID group must send a Neighbor Response message (SCSP type code 7) to the sender.

If SCSP is running over TCP/IP, as in the TRIP/SCSP architecture, the Time To Live (TTL) value of the Neighbor Discovery messages can be

[1] An IP multicast address has to be obtained for the TRIP/SCSP architecture.
[2] Type codes 1-5 are already in use.

controlled in order to keep track of the distance in IP hops to the destinations. This information can be used later by the unconfigured server as one of the input parameters to the decision process (decide the servers that will be DCSs). Thus, the first Neighbor Discovery message sent has $TTL = 1$, which means that it is only broadcasted in the local network. The server then sets a timer and waits for responses. All packets received within this interval are assigned the value $TTL = 1$. Once the timer expires, the TTL is increased and the server repeats the same steps. The timer value can be also updated according to the TTL, since responses from remote nodes are expected to take longer to come back to the sender. Both the TTL values and the timer values can be chosen freely by the administrator, although there must be a set of standard values to be used by default. When the TTL value reaches TTL_MAXIMUM_VALUE, the server stops sending packets. By then, it is supposed to have received one message from every server in SG_ID, providing TTL_MAXIMUM_VALUE is high enough.

6.5.3 Decision process

The Neighbor Response message contains the configuration information of the corresponding remote server. By receiving one of these packets from every node in the SG, the unconfigured server can gather enough information to run the decision process and choose a set of servers to connect.

As we stated earlier, every SCSP based architecture may have its own requirements. This implies that the criteria followed to select the peers during the decision process has to be system-dependent. Thus, we provide a configurable open mechanism that can be easily adapted to the requirements of every single system. The mechanism has two components:

- **Neighbor response message open format.** The Neighbor Response message contains the following information:

  1. Identity of the remote server that sends the Neighbor Response back to the unconfigured node.
  2. Neighbor_Weight. An integer value that specifies the weight the remote server gives to itself. This value represents the information that the remote server knows about itself.
  3. Identity of all the remote server's DCSs.
  4. Neighbor_Peer_Weight. For every DCS, an integer value specifying the weight assigned by the remote server to the DCS. This represents the information that the remote server knows about its peers.

  By receiving a Neighbor Response from every remote server, the unconfigured node knows both the identities of the servers in the SG and

the relationships between them. In other words, it gets a "picture" of the SG topology. The way in which the weights are calculated is completely system-dependent and must be chosen taking into account the requirements of the SCSP based architecture. Therefore, the Neighbor Response message provides an open way to exchange information about the SG members and the SG topology. The format of the Neighbor Discovery and Neighbor Response messages is illustrated in figure 6.3.

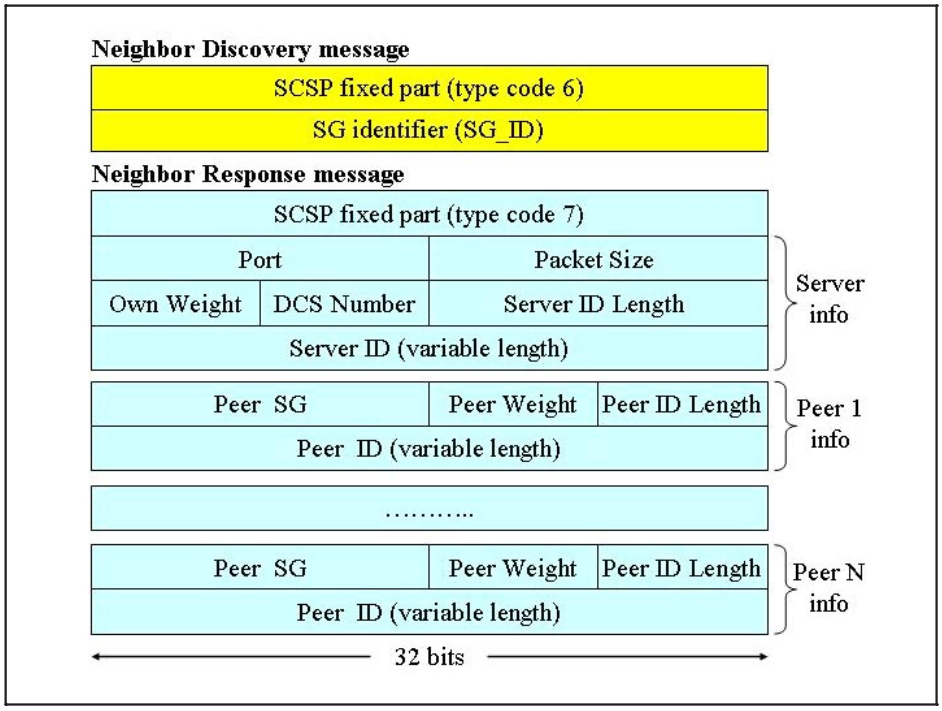

Figure 6.3: Neighbor Discovery and Neighbor Response message formats

- **Decision function.**

  The decision function is used to calculate a priority for every remote server present in the SG. It takes the following information as input parameters:

  1. Static_Neighbor_Weight values. The weights assigned by the unconfigured server to the remote servers. Before starting running, the unconfigured server might be provided by the administrator with some static priorities about the servers to be chosen.
  2. The Neighbor_Weight values received in the Neighbor Response messages.

3. The Neighbor_Peer_Weight values received in the Neighbor Response messages.
4. The TTL value calculated for the remote servers during the network discovery phase, providing the underlying transport layer supports this functionality.

All weights and TTL values are integer numbers. The function must process the information and assign a priority (also an integer number) to every remote server. The operations used to process the input parameters depend on the function itself, which is also system dependent.

This arrangement provides the auto-configuration module with a great deal of flexibility. For instance, the decision function can be calculated in such a way that it encourages the presence of star topologies within the SG. The weights sent in the Neighbor Response message can be based on any type of information: amount of traffic between peers, number of peers per server, etc. Furthermore, the distance in hops given by TTL values may be ignored by a specific decision function if geographic proximity is not a primary concern when selecting the peers.

6.5.4 SG connectivity calculation

When the decision process ends, every remote server must have been assigned an integer priority. The unconfigured server has then to calculate the number of DCSs to connect, which depends on the SG connectivity. Assuming that the SG is $k-connected$ before the server addition, we must ensure that at least the same connectivity value is maintained once the new node has been included into the SG structure.

We have developed a function that calculates the SG connectivity using graph theory algorithms. This function uses all the topological information received within the Neighbor Response messages to create a graph that represents the internal topology of the SG. If the SG connectivity turns out to be k, then the server must select k peers to maintain at least the same connectivity.

Since we have not been able to find an implementation of the vertex $k-connectivity$ in C programming language, we have developed our own function based on a free implementation of the maximum network flow algorithm. This implementation can be found in [37].

Servers with higher priorities are selected first. For instance, if the SG is 3-connected, the unconfigured server selects the three remote servers with the higher priorities to be DCSs.

6.5.5 Configuration file creation

The server creates a "hosts" file and adds the information about the selected peers to it. Thus, it can directly obtain the configuration information from this file if it goes down, which ensures that every server only needs to run the intra-domain auto-configuration procedure once. Both the port and packet size values are taken from the Neighbor Response messages received and included in the file. The SCSP related parameters are assigned default values.

6.5.6 Connection establishment

Once the configuration file has been created, the server has gathered enough information to run SCSP, so it starts sending Hello messages to the selected peers. Upon reception of one of these messages, the remote peer detects that it comes from an unknown server and runs a procedure to add the sender to the "hosts" file.

6.5.7 Certificate handling

If the architecture is making use of the SCSP security services provided by the security module, every server must possess the digital certificates of all its DCSs in order to have access to the corresponding public keys. Therefore, the new server has to obtain the certificates of its future peers and vice versa. Since SCSP is not responsible for dealing with certificates, they must be requested from the application layer.

Whenever the SCSP layer requires new certificates, a Certificate Request message is sent to the control interface. This message carries a set of IP addresses. The request message is retrieved by the application layer, which is responsible for obtaining the certificates of the nodes identified by the IP addresses. The way to perform this task is also system-dependent (i.e. the mechanism being used to store and distribute the certificates), but is usually done by sending the corresponding queries to a certificate server based on LDAP or DNSSEC. Once the certificates have been obtained, the application layer sends a Certificate Response to the control interface, indicating the file system paths where they have been stored.

In the TRIP/SCSP architecture, the application layer is represented by TRIP, that performs the certificate requests by querying the certificate server of the TRIP server authentication PKI described in section 5.2.

6.5.8 Memory consumption

The current SCSP implementation available at the lab, described in [35], was designed to minimize the amount of memory required in the SCSP

servers. This minimum memory space is needed for the Finite State Machine (FSM) allocation corresponding to each DCS. Following this criteria, we have also tried to design the auto-configuration module for minimum memory consumption.

The amount of memory required in the unconfigured server when the intra-SG auto-configuration procedure is executed can be easily obtained by examining the data structures used in the process. The Neighbor Discovery phase is the most critical in terms of memory consumption, since the unconfigured server must store all the information received in the Neighbor Response messages. The current implementation uses linked lists as the basic structure to store the information about remote servers. The Neighbor Response message contains information about the remote server and all its DCSs, as shown in figure 6.3, and this information is mapped into the memory structures, which are dynamically allocated. Assuming 4 bytes for integers and pointers, and two bytes for shorts, the expression that gives the amount of memory required during the Neighbor Discovery phase is:

$$16 + 60 * N + \sum_{i=1}^{N} 24 * P_i \; bytes \tag{6.1}$$

where N is the total number of remote servers that have sent a Neighbor Response message back to the unconfigured server, and P_i is the number of peers of remote server i.

For instance, if there are 50 servers in the SG with 3 peers per server on average, and all of them receive the Neighbor Discovery request, the unconfigured server is expected to receive 50 Neighbor Response messages. Applying equation 6.1, the total amount of memory required turns out to be:

$$16 + 60 * 50 + \sum_{i=1}^{50} 24 * 3 = 16 + 60 * 50 + 50 * 24 * 3 = 6616 \; bytes$$

We must remark that all this memory is freed once the auto-configuration process has been completed.

6.5.9 Intra-SG auto-configuration SDL model

We have developed a SDL model in order to clarify the operation of the intra-SG auto-configuration module. This model can be found in the Appendix B.

6.6 Inter-SG auto-configuration service

The purpose of this service is to automatically create new SGs to interconnect other existing SGs. It can be used in two different contexts: inter-area

and inter-domain. In the former one, an operator owns an ITAD which is divided into different areas. All servers in one area belong to the same SG. The information exchange between areas requires the establishment of additional inter-area SGs, which must contain at least one server from each area involved in the exchange. We will not impose any restriction on the number of areas represented in the inter-area SG nor to the number of servers of each area participating in it.

In the inter-domain case, we consider two or more ITADs belonging to different operators. The internal structures of the ITADs are irrelevant, i.e. they may consist of a single SG or be divided into separate areas (several SGs). The information exchange between operators is carried out by means of inter-domain SGs, whose components are representatives of the ITADs involved in the agreement. As in the previous case, we do not restrict the number of operators taking part in the inter-domain SG nor the number of representatives of each ITAD.

Although the contexts described are different, the auto-configuration requirements are basically the same for both of them, so our module only provides one common inter-SG auto-configuration service to be used in either situation.

The inter-SG procedure begins when a SCSP server receives the Create_SG signal from the application layer. This signal carries a set of SG identifiers representing the remote SGs that will participate in the new inter-area or inter-domain SG. The way in which these identifiers are obtained is responsibility of the application layer. For instance, within the TRIP/SCSP architecture there are two possible mechanisms to obtain them, depending on whether we have an inter-area or an inter-domain context:

- For inter-area, the TRIP layer contains a table with information about all the areas present in the ITAD and their corresponding SGs. This table can be used to select the set of SG identifiers carried in the Create_SG signal depending on the topology requirements of the ITAD.
- For the inter-domain context, the TRIP layer contains a table with information about remote ITADs and their corresponding SGs (or single SG if the operator has not defined areas within the domain). The TRIP server must first decide the remote ITADs that will be involved in the new agreement and then look up their corresponding SG identifiers in the table.

The SCSP server that receives the Create_SG signal acts as a master server in the inter-SG auto-configuration procedure.

6.6.1 Gateway Discovery phase

Upon receiving the Create_SG signal, the master server must find other remote servers belonging to the SGs that will participate in the new SG,

including its own SG[3]. We will refer to these servers as gateways, since they act as "gateways" between different SGs. The master server starts sending Gateway Discovery messages which include the SG identifiers obtained from the application layer. The identifier of the master server's SG is appended as well. As in the Neighbor Discovery phase described in section 6.5.2, messages are encapsulated in transport level multicast messages and sent to the SCSP auto-configuration multicast address. If SCSP is running over TCP/IP, the TTL of the messages may be increased in the way described in section 6.5.2 to keep track of the distance in hops to the gateway candidates.

Remote servers belonging to any of the SG identifiers carried by the Gateway Discovery message must process the incoming message. The new SG will be used to share information between all the SGs whose identifiers are present in the message received, so before replying to this message the gateway has to make sure that it is authorized to exchange information with these SGs. Again, the application layer is responsible for this task. The SCSP layer creates a new Authorize_SG_Request message which includes all the SG identifiers present in the Neighbor Discovery message, and sends it to the interface. The application layer processes this message and checks if the local SG has agreements with all the other SGs. The response is sent back to the SCSP layer by means of an Authorize_SG_Response. If this response is affirmative, then the gateway sends a Gateway Response message back to the master server. Otherwise the incoming Neighbor Discovery message is ignored and no response is generated.

We must remark that this security procedure is absolutely required to maintain the security of the overall architecture. It prevents the establishment of unauthorized inter-area and inter-domain SGs and ensures that an area (or domain) is exchanging information only with those areas (or domains) with which previous exchange agreements have been established.

The Gateway Discovery phase ends when the master server has collected all the Gateway Response messages from the remote servers.

6.6.2 Decision process

As in the intra-SG auto-configuration service, we provide an open mechanism that can be customized in order to match the requirements of different architectures. The mechanism has two components:

- **Gateway Response message open format.** The Response message contains the following information:

    1. Identity of the gateway that sends the Gateway Response back to the master server.

[3]This can be useful to include in the new SG more servers from the master server's SG.

2. Gateway_Weight. An integer value that represents the "willingness" of the remote server to become a gateway and to be included in the new SG. The way in which this value is calculated depends on the specific requirements of the SCSP based architecture.
3. SG_Identifier_List. The identifiers of all the SGs the remote gateway belongs to. The master server needs this information in order to choose a free SG identifier for the new group. Choosing an identifier which is not being used by any of the selected gateways is a crucial task to ensure the proper operation of the protocol and to prevent undesired behaviours.

The format of the Gateway Discovery and Gateway Response messages is illustrated in figure 6.4. Gateways can also include a port number and a packet size value in the message, so that these parameters can be negotiated during the establishment of the new SG.

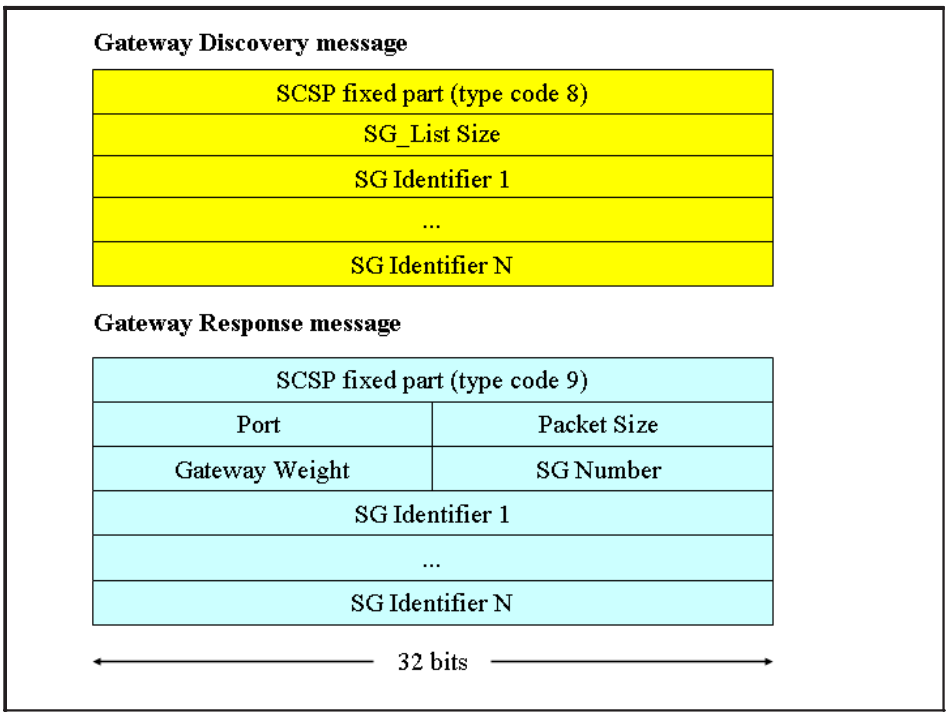

Figure 6.4: Gayeway Discovery and Gateway Response message formats

- **Decision function.** The master server then runs its decision function in order to compute a priority for every remote gateway. Again, this function is completely system-dependent and must be configured considering the system requirements. The following information is taken as input parameters:

1. Static_Gateway_Weight values. The master server may be given some preferences about the identities of the gateways to be included in the new SG before running the inter-SG procedure. This represents the information the master server knows about the gateways prior to the execution of the procedure, and may be optionally considered when selecting gateways.
2. The Gateway_Weigth values received in the Gateway Response messages.
3. The SG_Identifier_List lists received in the Gateway Response messages.
4. Distance in hops to the remote gateways (providing the TTL mechanism is in use).

6.6.3 Selection of the SG identifier

The master server has to select a SG identifier for the new SG. SCSP reserves 16 bits to store SG identifiers, which means that there might be a maximum of 65.536 different SGs in a system.

If we assume that SG identifiers can not be reused, the master server should have knowledge about all the identifiers being used in the system in order to select a free one. This task is completely unfeasible, since it requires a parallel mechanism to distribute the used SG identifiers throughout the entire architecture.

Instead, we divide the total range of identifiers into two groups. The first one encompasses identifiers 0-59.999, and is reserved for conventional SGs, i.e. not inter-area nor inter-domain SGs. Any SG used to replicate the information within an area or an ITAD is identified by an integer taken from this interval, and these identifiers can not be reused.

The second group encompasses identifiers 60.000-65.535 and is reserved for inter-area and inter-domain SGs. These identifiers can be reused providing we ensure that one server does not belong to two SGs with the same SG identifier.

Since the Gateway Response message includes the list of SG identifiers the remote server belongs to, the master server knows all the SGs where the selected gateways are already replicating. The lowest unused identifier is chosen for the new SG. For instance, if none of the gateways selected to take part in the new SG (including the master server) belongs to any inter-area or inter-domain SG, the identifier 60.000 will be chosen for the SG. On the other hand, if any of the gateways is already using the SG 60.000, then the master server will choose the identifier 60.001. The master server applies this mechanism in order to ensure that SG identifiers are not repeated and that the lowest possible identifier is always chosen. Therefore, identifiers are locally unique. Even though they can be reused throughout the system, the

proper operation of the architecture is guaranteed and servers can not be replicating in two SGs with the same identifier.

The auto-configuration module is not responsible for allocation of identifiers 0-59.999 to ITADs or areas. This task is left to the administrator of the system, who must ensure that they are unique.

6.6.4 Topology limitations and Hello Forward extension

We must notice that so far the inter-area or inter-domain SGs created as a result of the inter-SG auto-configuration procedure would always have star topologies, with the master server located in the centre of the star. The master server is directly connected to all the remote gateways, while the gateways are only connected to the master server. This is due to the fact that the master is the only server which knows the identities of all the members of the new SG. As we explained in section 6.5.1, star topologies do not perform well for high update rates, because the intermediate node quickly saturates. Furthermore, the star topology is rather unreliable in terms of fault tolerance, since it is a 1-connected graph. This means that the failure of any of the servers in the SG makes the entire group to become disconnected as well.

In order to address these limitations, we have designed a mechanism that allows the master server to manage the topology of the new SG. The mechanism consists of a new SCSP extension, called Hello Forward extension, optionally carried by Hello messages. Once the master server has selected a set of gateways to be included in the new SG, it runs a function to choose a suitable topology for the group. The primary concern when selecting the topology is reliability. If we want the new SG to operate properly when k-1 nodes go down simultaneously, then the graph representing the SG must be at least k-connected, and the topology has to be configured to match this connectivity.

The Hello Forward extension can be used by the master server to inform the remote gateways about their peers in the group and to control the topology. The extension tells gateways about the identities of other gateways that must be directly connected to them so that the topology of the new SG matches the desired connectivity. Thus, after running the topology management function the master server knows which gateways are going to be contacted directly and the ones which are going to be contacted indirectly through other gateways. When a gateway receives the first Hello message from the master server, it checks if the Hello Forward extension is present in the message. This extension carries one ore more server addresses identifying those servers that must become peers of the gateway. With this information, the gateway can add the servers to its configuration file and start sending Hello messages to them. The structure of the new Hello Forward extension is depicted in figure 6.5.

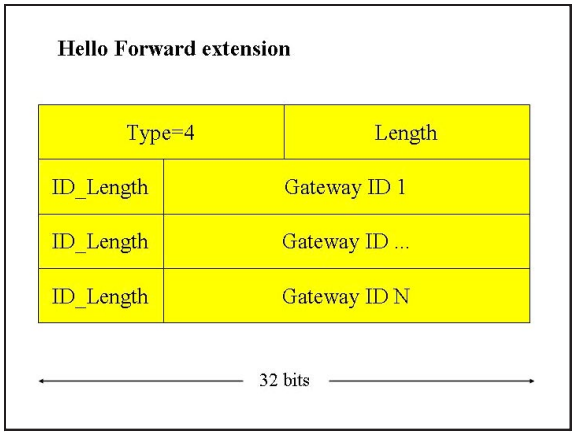

Figure 6.5: Hello Forward extension format

Let us consider a simple example in order to clarify the topology management mechanism presented in this section and the use of the Hello Forward extension. Figure 6.6 shows a system using SCSP as the underlying replication layer. There are three domains and a new inter-domain SG must be established so that the three domains can share information with each other. The master server (server 4) runs the Gateway Discovery and receives Gateway Response messages from servers 1-3 and 5-10. After running the decision process, servers 2, 5, 6 and 8 are assigned the higher priorities. Assuming that the master server selects these servers to take part in the new SG, and that it does not use the Hello Forward extension mechanism, direct connections are established between the master server and all the selected gateways. The resulting SG presents a star topology, with the master server acting as an intermediate server. The main drawback of this approach, depicted in figure 6.6, is that we have no control over the SG topology. The k-connectivity of the resulting graph is always 1 and any single server failure may disconnect the graph.

Figure 6.7 depicts the same scenario when topology management is used. The master server runs the decision function to calculate the gateway priorities, as in the previous case. However, now we impose the additional requirement that the resulting inter-domain SG (which will include servers 2, 5, 6 and 8) is 3-connected, so that it can tolerate a maximum of two simultaneous server failures. The master server runs the topology management function in order to find a topology that matches this requirement. By using the Hello Forward extension, the selected gateways are informed about their peers in the new SG. For instance, upon receiving the Hello message from the master, server 5 unpacks the Hello Forward extension carried in the message and finds out that it must set up a direct connection with server 6.

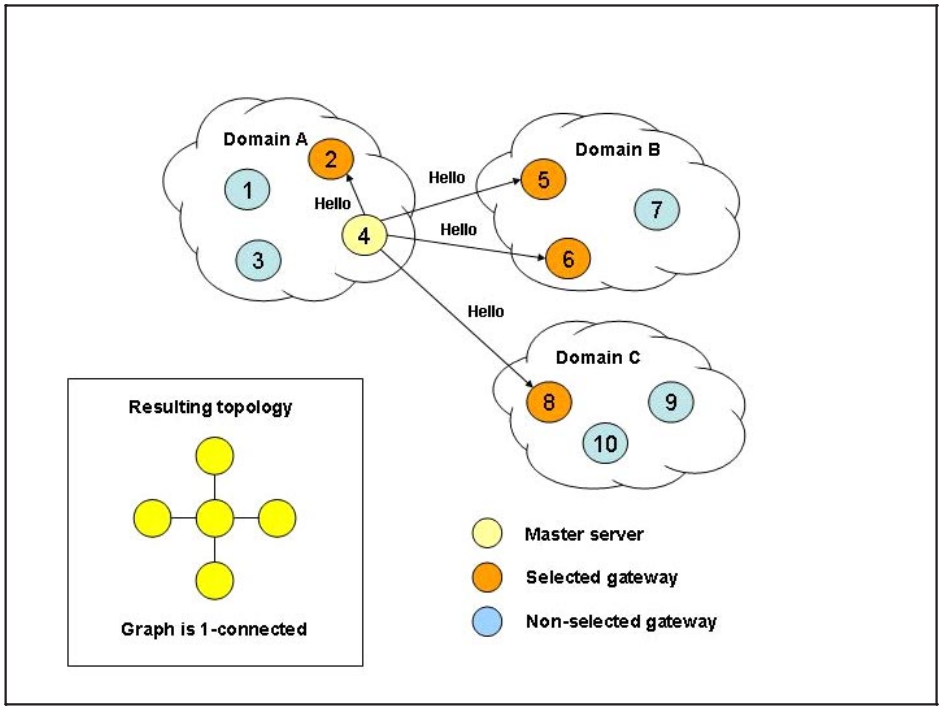

Figure 6.6: Hello Forward extension usage example (1)

Figure 6.7 shows that in this case the resulting graph is the 5-element wheel graph, which is 3-connected and matches the reliability requirements. This structure tolerates up to 2 simultaneous server failures without being disconnected.

The topology management facilities provided by the Hello Forward extension add a considerable complexity to the inter-SG auto-configuration procedure, but they are absolutely necessary if we want to have fine control over the topologies of the inter-area or inter-domain SGs being created. The use of this mechanism is optional, and it can be omitted in those systems where star topologies are good enough to fulfil the requirements.

6.6.5 Connection establishment and configuration file update

The master server creates an entry for the new SG in the configuration file and adds the information about the selected gateways to it. This guaranties that the new SG is registered in the "hosts" file of the master server and that it is automatically restored after a system failure. If the system uses topology management, then the master server only adds to the file the information about those servers which are going to be contacted directly, and fills the Hello messages with the Hello Forward extensions needed to achieve the

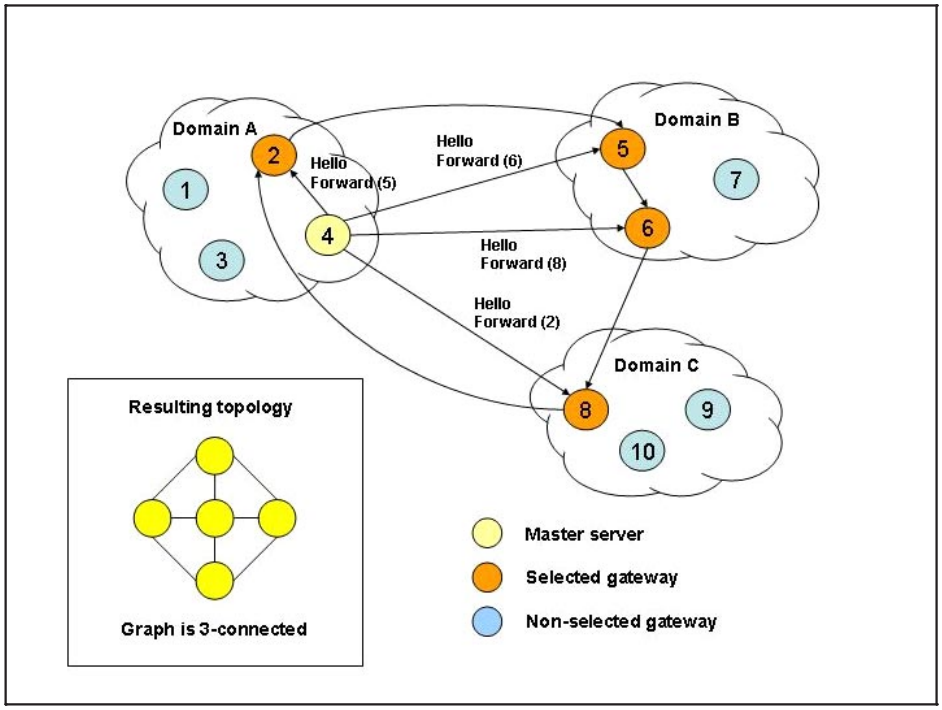

Figure 6.7: Hello Forward extension usage example (2)

desired SG topology. The master server then starts sending Hello messages to the selected gateways, which detect that the sender is unknown and run a procedure to update their configuration files. The gateway finds out the identifier of the new SG by inspecting the Hello message received, and creates a new entry for that SG in the configuration file. The information about the master server is then added to the file as well. If the message carries a Hello Forward extension, it means that the master server is using topology management. The gateway must add the servers present in the extension to the configuration file and establish connections with them.

6.6.6 Certificate handling

If SCSP is using the security services provided by the security module, servers involved in the new SG have to obtain the certificates of their future DCSs. The procedure described in section 6.5.7 for the intra-SG procedure is also applicable here to retrieve the required certificates.

6.6.7 Memory consumption

The master server must store all the information received in the Gateway Response messages, because this information is needed to run the decision

function. As in the intra-SG case, the information is kept in memory structures that are dynamically allocated. The expression that calculates the total amount of bytes required in the master server is (again assuming 4 bytes for integers and pointers and 2 bytes for shorts):

$$12 + 48 * N + \sum_{i=1}^{N} 8 * S_i \; bytes \tag{6.2}$$

where N is the total number of remote gateways that have sent a Gateway Response message back to the master server, and S_i is the number of SGs in which server i is already replicating.

As an example, lets assume that the master server receives a response from 50 remote gateways and that they belong to three SGs on average. Then, the amount of memory required to store all the received information in the master server is:

$$12 + 48 * 50 + \sum_{i=1}^{50} 8 * 3 = 12 + 48 * 50 + 50 * 8 * 3 = 3612 \; bytes$$

6.6.8 Inter-SG auto-configuration SDL model

The SDL model for the inter-SG auto-configuration procedure is presented in the Appendix B.

6.7 Definition of profiles for the auto-configuration module

The auto-configuration module proposed in this chapter provides SCSP based architectures with an open and customizable auto-configuration mechanism. This means that it can not be directly used by a system, since the way in which the different priorities and weights are calculated, as well as the way in which those values are processed by the decision functions, is not defined. In order to be applicable, the module has to be completed with an auto-configuration profile. Every SCSP based system has to define its own profile taking into account the requirements of the architecture and the nature of the information which is going to be replicated by SCSP. This nature determines the update rates that SCSP must support.

Tables 6.1 and 6.2 summarize the parameters that must be defined by any auto-configuration profile. These tables can be used as guidelines when designing new profiles.

Parameter	**Description**
Static_Neighbor_Weight values	Decide whether these values are going to be used by the system and the way in which the unconfigured server becomes aware of them
Neighbor_Weight values	Mechanism used by a remote server to calculate its own weight.
Neighbor_Peer_Weight values	Mechanism used by a remote server to compute weights for its directly connected peers.
Decision function	Decide the function that transforms all the inputs to the decision function into a single value for each server that represents the final priority of the server. Also decide whether the TTL values (if available) are taken into account when computing the priorities.

Table 6.1: System-dependent parameters in the intra-SG auto-configuration procedure

Parameter	**Description**
Static_Gateway_Weight values	Define whether these values are going to be used by the system and the way in which the master server becomes aware of them.
Gateway_Weight values	Mechanism used by a remote gateway to calculate its own weight.
Decision function	Decide the function that transforms all the inputs to the decision function into a single value for each server that represents the final priority of the server. Also decide whether the TTL values (if available) are taken into account when computing the priorities.
Topology management	Decide if topology management is required in the system. Specify the mechanism to select suitable topologies for the new SGs and define how the Hello Forward extensions are used to deploy the topologies.

Table 6.2: System-dependent parameters in the inter-SG auto-configuration procedure

6.8 Auto-configuration profile for the TRIP/SCSP architecture

6.8.1 Characterizing the TRIP/SCSP architecture

The main parameter we must consider in order to successfully characterize a SCSP-based architecture is the update rate generated by the application layer, since this determines the amount of information sent to the SCSP layer to be replicated. In general, TRIP generates very low update rates. For instance, following the assumptions made in [31], when TRIP is used to implement Local Number Portability (LNP), the system must tolerate between 0.5 and 4 updates per second.

6.8.2 Intra-SG profile

This low update rate means that our system can be mainly based on star topologies, since a server can support several DCSs without getting saturated. The presence of star topologies will help to reduce the replication time for the updates (updates must traverse fewer hops to reach their destinations), and the overall replication time for the SG will be reduced.

Encouraging star topologies implies that the unconfigured servers must always connect to those neighbours with the highest number of DCSs within the SG under consideration. However, the following factors have to be taken into account when running the decision function:

- The maximum number of DCSs supported by a server. Although we encourage star topologies, an excessive number of DCSs worsens the performance of the server. The reason is that every peer requires the establishment and maintenance of several finite state machines and data structures which consume memory and CPU capacity. Therefore, we have to limit the maximum number of DCSs per LS, even though the amount of updates generated by them is not critical. The Neighbor_Weight value is calculated as the maximum number of DCSs supported by the remote server. This value can be manually loaded into the server when it is first set up, and is calculated considering the features of the machine (amount of memory, CPU speed, etc). Thus, the more powerful a server is the more DCSs it can support.

- The amount of updates a remote server is receiving from its DCSs. Even though a server may be able to support more DCSs (because the maximum number of DCSs, specified by the Neighbor_Weight value has not been reached yet), we need to monitor the amount of updates it is receiving in order to ensure that the server will not saturate if a new DCS is added. Our current SCSP implementation includes support for the SCSP Management Information Base (MIB)

described in [38]. This allows the network administrator to remotely manage SCSP servers. The MIB definition consists of three tables: scspServerGroupTable, scspLSTable and scspDCSTable. The scspDCSTable contains information associated with the opened session between the LS and the corresponding DCS, and keeps track of statistics about message exchanges. The LS maintains a separate scspDCSTable for every DCS. One of the entries of the scspDCSTable is scspDCSCSURequestIn, which registers the number of CSU Requests received from the DCS. This value can be used to calculate the update rate generated by the peer. The scspDCSCSURequestIn value is inspected every Time_Interval seconds. The update rate (in updates per second) of DCS i in Time_Interval k is computed by applying the following expression:

$$Update\_Rate_i(k) = \frac{CSU\_Requests_i(k)}{Time\_Interval\ seconds}$$

$$= \frac{scspDCSCSURequestIn_i(k) - scspDCSCSURequestIn_i(k-1)}{Time\_Interval\ seconds} \tag{6.3}$$

Every Time_Interval seconds the LS must update the rates of all its DCSs. The value of Time_Interval must be carefully chosen. A small value will cause the update rates to be inaccurate. However, a big value may cause them to be useless.

The Neighbor_Peer_Weight of peer is calculated as the update rate (in bytes per second and taking the integer part) generated by the peer.

- The distance in IP hops to the remote peer. Since the TRIP/SCSP architecture runs over TCP/UDP, the TTL mechanism is available and can be used to estimate the topological distance to the server. Communications with closer servers are expected to be more reliable and the delay due to router processing grows with the number of IP hops. Thus, remote servers with low TTL are more likely to be chosen.

We assume that the unconfigured server has no prior knowledge about the neighbours present in the SG, so the Static_Neighbor_Weight values are not used within this auto-configuration profile.

Decision function

The final priority of a server is calculated as the number of DCSs the server has in the SG under consideration. This value is computed by inspecting the Neighbor Response message received from the remote server. Following this criteria, nodes with more peers will be assigned higher priorities. However,

the priority of a server is set to zero if any of the following conditions is matched:

- The server has already reached its DCS limit, i.e. the DCS Number value (received in the Neighbor Response message, see figure 6.3) equals the Neighbor_Weight value of the node.
- The server is being subjected to heavy traffic loads. The total current load supported by the node is computed as the addition of all the Neighbor_DCS_Weight values of the DCSs. If the resulting rate exceeds a threshold, then the server is close to saturation (it is receiving too many updates from its existing DCSs), and new DCSs can not be supported.

In case two servers are assigned the same priority, the one with a lower TTL value is selected first.

6.8.3 Inter-SG profile

Unlike conventional SGs, the topology of the inter-area or inter-domain SGs can be completely managed by the master server using the topology management function. Therefore, we can decide which topology the new SG is going to have, and there is no need to encourage star topologies. Based on this fact, the proposed inter-SG profile for the TRIP/SCSP architecture can be summarized as follows:

- The Gateway_Weight values are computed as the total number of DCSs connected to the gateway.
- We assume that the master server has no prior knowledge about the other gateways, so the Static_Gateway_Weight values are not used within this auto-configuration profile.
- The TTL mechanism is used to keep track of the distance in IP hops from the master server to the gateways.
- Two gateways from each area (or ITAD) will be selected. This ensures the presence of two representatives from every area (or ITAD) in the new SG and guarantees the replication in all the areas (or ITADs) when an arbitrary node goes down.
- The decision function computes the final priority for a gateway as the sum of the TTL value and the Gateway_Weight value. In this case, gateways with lower priorities will be selected first. Thus, we tend to select those servers with low processing loads (few DCSs) and short distances to the master server (low TTL values).

- The topology of the new SGs is always the wheel, which is a 3-connected graph. To achieve this topology, all the gateways become DCSs of the master server. In addition, every gateway receives a Hello Forward extension from the master server with the address of another gateway.

To sum up, servers are organized in wheel arrangements, with the master server located in the middle of the wheel. With this internal disposition (a 3-connected graph), new SGs are able to tolerate up to 2 simultaneous failures without being disconnected. However, if the failures affect two servers of the same area (or ITAD), then that area (or ITAD) will be isolated until one of them is recovered.

6.9 Auto-recovery service for network load optimization

This section presents an alternative auto-configuration mechanism for SCSP based systems where network load optimization is a primary concern. The intra-SG and inter-SG services described in this chapter provide a complete auto-configuration mechanism for SCSP based systems. However, they make use of the k-connectivity factor of the SG to guarantee the SG graph connectivity. As we know, this approach establishes that the undirected graph representing an SG must be k-connected if we need to guarantee connectivity when k-1 nodes fail simultaneously. Thus, giving the k-connectivity a value of 3 ensures that the SG can tolerate up to 2 arbitrary node failures.

The main disadvantage of this approach is that it introduces redundancy in the information exchanged within the SG. The presence of redundant paths between servers ensures that the SG can tolerate failures without any dynamic topology reconfiguration, but it also means that the same information is sent through different paths. This produces an unnecessary network load.

Although most systems will be able to accept this extra traffic in the network, there might be some special systems with severe network traffic constrictions. This section describes an alternative auto-configuration mechanism for this kind of systems.

6.9.1 Definition of a minimum-traffic SCSP system

We define a minimum-traffic system as a SCSP based system with the following features:

- The system must be optimized in terms of network load.
- As a consequence of the previous statement, the graphs representing the SGs in the system must be 1-connected.

6.9.2 Auto-recovery service description

Since 1-connected graphs are likely to be disconnected with a single node failure, we need to provide an additional mechanism to ensure that replication within the SG keeps working properly. We call this new mechanism auto-recovery procedure.

The auto-recovery procedure detects the failure of a node and automatically reconfigures the SG topology to ensure that the corresponding graph remains connected. This is achieved by establishing new temporary connections between servers. Once the server that failed is recovered, the temporary relationships are removed and the topology of the SG returns to its original state.

We will apply our study to the two basic 1-connected topologies: inline and star. These topologies are illustrated in figure 6.8. We must notice that this figure assumes that servers have a limited capacity in the star case. When the server located in the center of the star reaches its maximum number of DCSs, another server must be chosen as the center of a new star. This is the reason why we present more than one star.

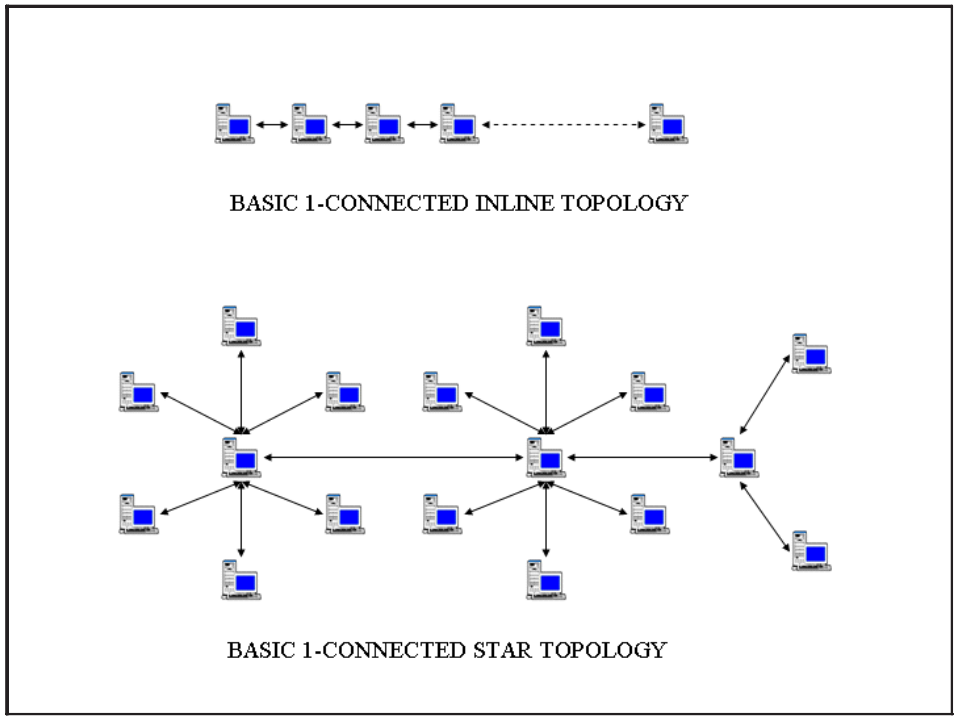

Figure 6.8: 1-connected inline and star topologies

The auto-recovery procedure involves the following tasks:

- Maintenance of up-to-date topology information. Every server located

within the SG must have up-to-date information about the state of the SG topology. This could be achieved by forcing all the servers in the SG to periodically flood Neighbor Response messages using the SCSP multicast address for auto-configuration. This mechanism would allow servers to advertise information about their DCSs at regular intervals. By receiving and processing all the flooded messages, a server gathers enough information to have a picture of the SG topology. However, flooding mechanisms always represent an extra load for the network. Since the auto-recovery procedure is designed for minimum-traffic systems, where network load optimization is a primary concern, the solution based on flooding must be discarded.

In order to distribute up-to-date topology information within the SG, we will follow an alternative approach that makes a better use of the available bandwidth. When the system starts running, every server floods a Neighbor Response message within their SGs. This provides all servers with information about the initial SG topology. After that, only the updates must be advertised. There are two basic updates:

- When a new server joins the SG (using the intra-SG service), it floods a Neighbor Response message within the SG to advertise its information. The other servers receive this message and register the incoming node.
- When a new SG is created using the inter-SG auto-configuration service, all the servers taking part in the SG must advertise their information by flooding a message within the SG.

The distribution of reliable topology information in the system is crucial for the correct operation of the auto-recovery procedure. All the actions taken by servers when a failure occurs will be based on this information, so it must be as accurate and consistent as possible.

- Failure detection. The DCSs of the server that goes down must become aware of the failure. This task is performed by the Hello Finite State Machine (HFSM) of the SCSP, which is responsible for monitoring the state of the connection between the LS and the DCS. Once the connection has been established, peers keep exchanging Hello messages to ensure that the connection remains active.

  As explained in [6], the HelloInterval parameter determines the time in seconds between sending of consecutive Hello messages. If the LS does not receive a Hello message from the DCS in HelloInterval seconds, then the message is late. After HelloInterval*DeadFactor seconds (DeadFactor being another SCSP parameter, an integer number), the DCS is considered to be inoperative.

Thus, servers can use this mechanism to check the connection and to decide when a DCS has gone down. The values of HelloInterval and DeadFactor must be carefully chosen. A big value will make the mechanism useless, since the LS will only be aware of the failure long after the peer has actually gone down. A too small value would generate an excessive amount of traffic in the network.

- The auto-recovery function. This function is run by a server when a failure in the SG is detected. The set of actions to be taken depends on the base topology being used (inline or star) and the position occupied by the server that breaks down.

As we have explained, the auto-recovery procedure assumes that the SGs within our system have 1-connected topologies. Thus, we must first define new auto-configuration profiles for the intra-SG and inter-SG auto-configuration services so that this assumption is fulfilled. By defining a proper intra-SG profile we ensure that new nodes are added to the existing SGs in a proper way. For instance, if the SG has an inline 1-connected topology, then new nodes must always connect to one of the ends of the line in order to respect the topology. Likewise, we must define the inter-SG profile so that the dynamically established SGs have always 1-connected (either inline or star, depending on our requirements) graphs.

For brevity, we do not include the profiles here, but they can be easily generated by following the guidelines given in tables 6.1 and 6.2.

The following sections present the auto-recovery functions for the inline and star topologies.

6.9.3 Auto-recovery function for the inline topology

When a server detects the failure of a DCS, it first floods a Neighbor Response message to advertise the change detected in the topology. This message does not include any information about the peer that has gone down, allowing the rest of the servers to become aware of the failure and to update their information about the SG topology.

Upon receiving this update, all the servers present in the SG must determine whether the failure affects an end node or an intermediate node. This can be easily done by inspecting the topology information of the SG. As we stated earlier, at this point it is crucial that all servers have access to up-to-date information about the current SG topology. The two possible situations are depicted in figure 6.9. A failure in an end node does not disconnect the SG, so no further actions have to be taken. Servers only wait for the inoperative node to recover.

A failure in an intermediate node requires a more complex processing, since it causes the SG to become disconnected. The rest of the servers of the SG must reconfigure the SG topology by establishing the required

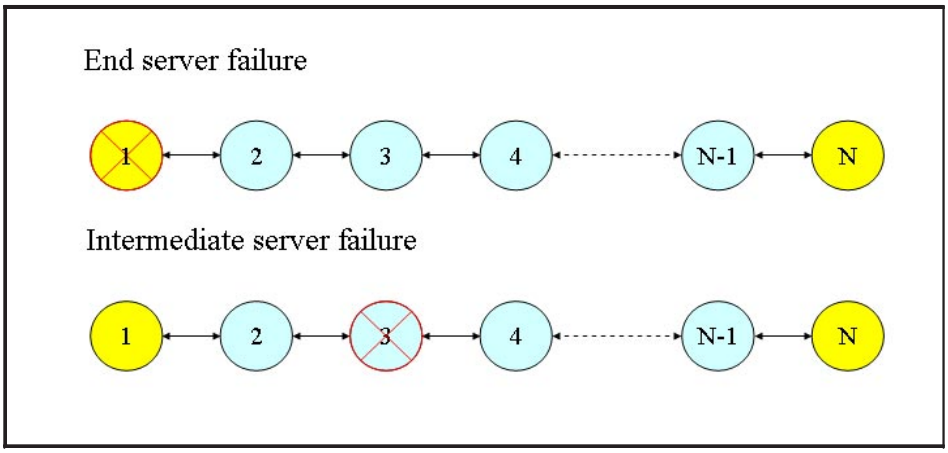

Figure 6.9: Possible failures in the inline topology

temporary connections. In an inline arrangement, the intermediate server that breaks down always has 2 DCSs. The first DCS that detects the failure must identify the other DCS of the node that failed and set up a temporary connection with it.

A temporary connection must be always associated with a node failure. As soon as the node that failed is recovered, it must perform the following actions:

- Send a multicast Neighbor Response message to the SG to advertise that it is fully operative again. Upon receipt of this message, all the servers of the SG will update their topology information to register the change. The temporary connection associated with the failure must be removed as well, since it is no longer needed.
- Send a Neighbor Discovery message to the SG. The reception of this message will force the other servers present in the SG to reply with the corresponding Neighbor Response message. This allows the recovered server to update its topology information.

Figure 6.10 shows a basic example of the auto-recovery function for the inline topology. Server 3 is not likely to be using the same HelloInterval and DeadFactor values for its connections with servers 2 and 4. This means that they will not become aware of the failure at the same time. The DCS that first notices the change (server 4 in figure 6.10) is always responsible for sending the Neighbor Response message and initiating the communication with the other DCS (server 2) to set up the temporary connection.

The association of temporary connections with server failures enables the system to cope with more than one failure at the same time. Figure 6.11 shows a more complex example with two simultaneous server failures.

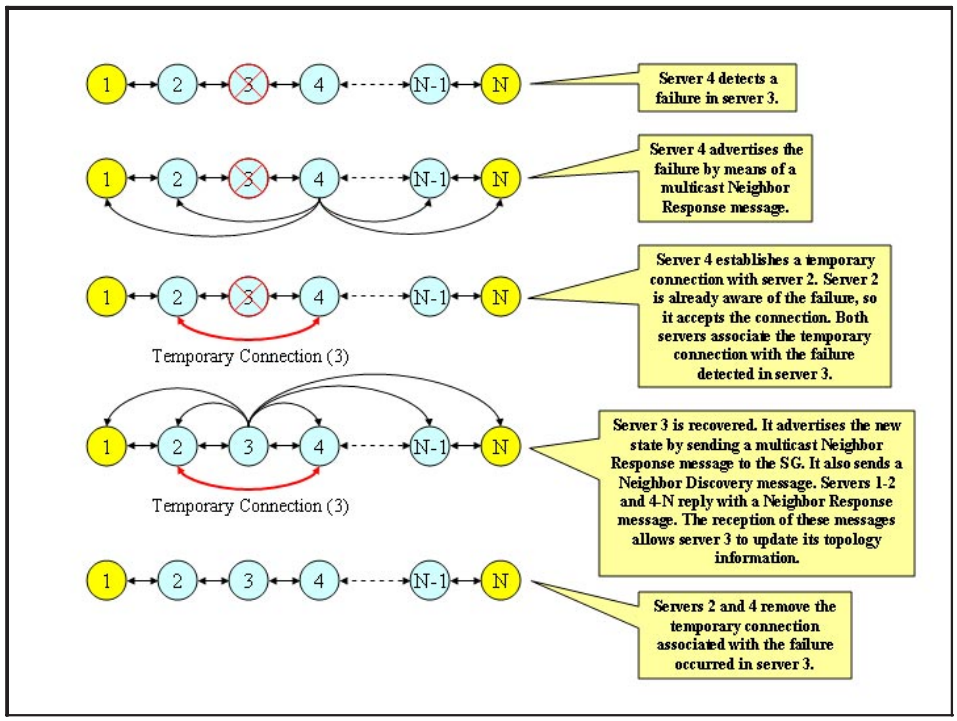

Figure 6.10: Basic auto-recovery example for the inline topology

The establishment of temporary connections must take place in a secure way. Thus, the pair of servers involved in the connection must get the digital certificates of each other. This will enable them to perform the peer authentication procedure during the Hello phase.

6.9.4 Auto-recovery function for the star topology

The auto-recovery function in this case is very similar to the one presented for the inline topology. There are also two possible failures, which are depicted in figure 6.12. In addition to the server ID, a server is also represented by its maximum DCS capacity, which specifies the maximum number of connections the server can accept. This value is calculated considering the resources available at the machine (memory, CPU, etc). The example in figure 6.12 shows three servers able to act as intermediate nodes (servers 3, 6 and 9). For simplicity, we consider that the rest of the servers can only accept one connection, which means that they can only act as end servers. We must notice that the topology depicted in figure 6.12 is in fact a combination of a simple star topology (one single star) and the inline topology. The intermediate nodes located in the middle of the stars are organized following the inline arrangement. This feature will allow us to reuse here the auto-recovery function developed for the inline case.

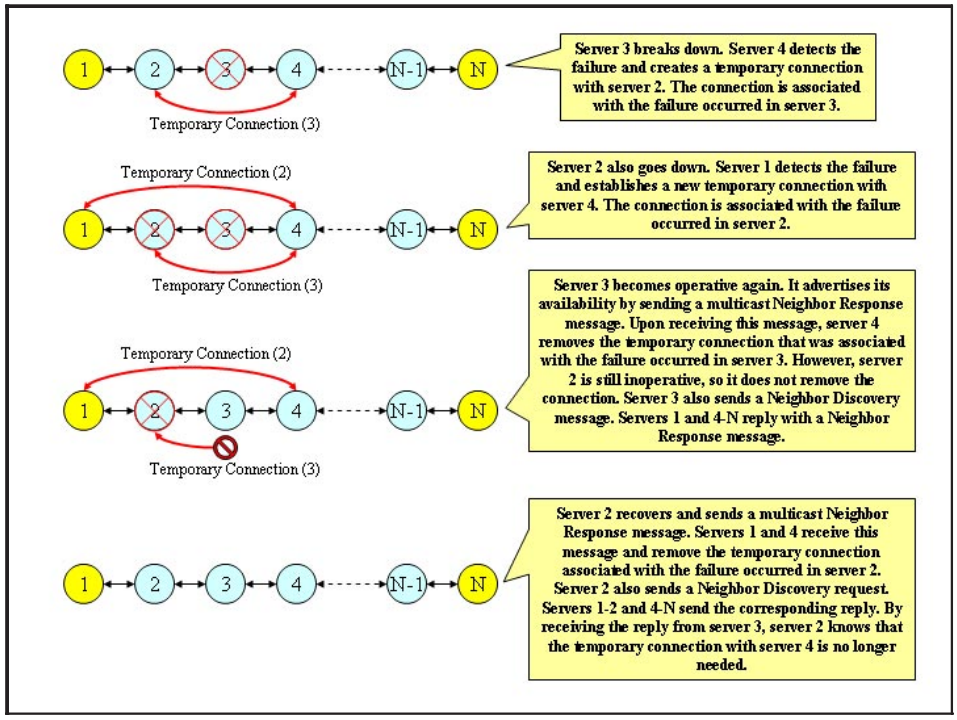

Figure 6.11: Advanced auto-recovery example for the inline topology

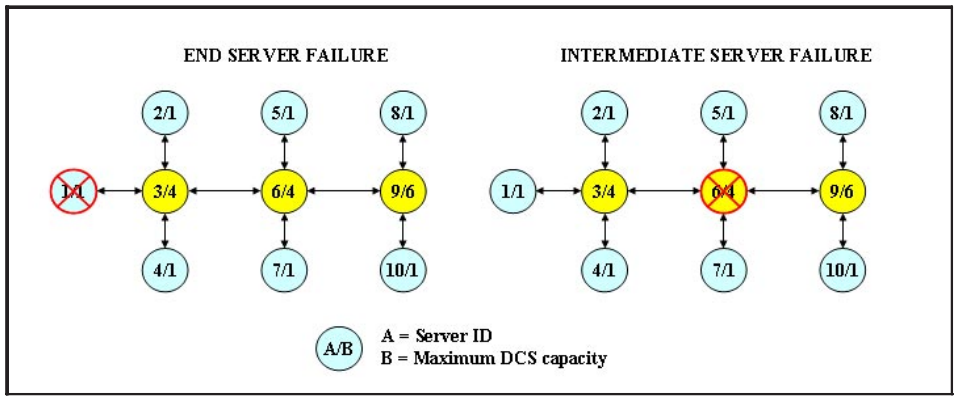

Figure 6.12: Possible failures in the star topology

To have a better understanding about how the auto-recovery function for the star case works, we must first briefly explain the intra-SG profile needed to achieve this kind of arrangement. When a new node wants to join a 1-connected SG based on the star topology, like the one shown in figure 6.12, it runs the intra-SG auto-configuration procedure. The policy specifies that the neighbor with a higher DCS capacity will be selected first, providing it has not reached its maximum capacity. This server will be called the active server of the SG. Thus, in our example, an incoming node will select server 9 to connect, since it is the active server (it has the higher DCS capacity (6) and it has not reached the limit yet, because there are only 3 servers connected to it).

The end server failure is again the easiest to deal with. The first server that becomes aware of the failure sends a multicast Neighbor Response message to advertise the change. An end server failure does not trigger further actions. The rest of the servers in the SG only wait for the node to recover.

If an intermediate server failure is detected, i.e. the failure has affected a node located in the middle of the star (server 6 in the example of figure 6.12), then the rest of the servers of the SG will react depending on their location within the topology:

- End servers which are not directly connected to the inoperative one (servers 2,4,8 and 10 in figure 6.12) do not perform any action.
- The rest of the intermediate servers present in the SG (servers 3 and 9 in figure 6.12) will behave as in the inline case, running the auto-recovery function for the inline case described in the previous section. End servers are ignored by intermediate servers when running this function.
- End servers which are directly connected to the inoperative one (servers 5 and 7 in figure 6.12 must behave as if they were new unconfigured nodes trying to join the SG. This means that they have to run the intra-SG procedure to find the active server and connect to it.

Figure 6.13 shows the resulting temporary connections for the intermediate server failure depicted in figure 6.12. We can see how servers 5 and 7 connect to server 9 after running the intra-SG auto-configuration procedure, since it is the active server.

As in the inline case, all the temporary connections created as a result of a server failure must be associated with the failure, so that they can be removed once the server in question has recovered.

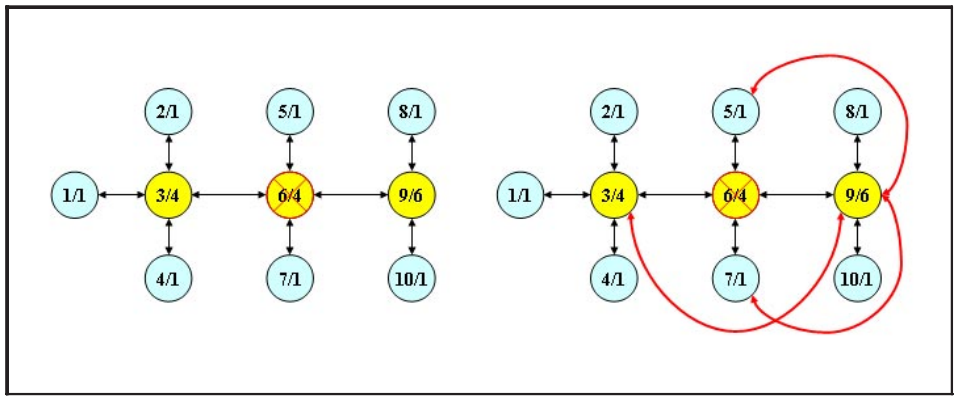

Figure 6.13: Auto-recovery example for the star topology

Chapter 7

Conclusions and future work

7.1 Conclusions

The first goal of this thesis has been to upgrade the previous version of the SCSP available at the laboratory. After re-implementing all the internal buffers using the gdbm technology, we carried out some tests to measure the performance of the new version. Results show that the replication speed has been considerably improved. Furthermore, the enhanced prototype has a linear response to the growth in the number of initial entries. Since the former version did not have such a linear response, it is rather difficult to establish a direct comparison between them in terms of speed. The improvement factor defined in chapter 4 can be used to perform the comparison, but it varies depending on the database size. This factor shows that the enhanced version is about 60 times faster when databases of 100.000 records are used.

The packet loss problem detected in the former version has been solved by eliminating all the points in the source code where packets were being sent in bursts to the network.

Despite all these improvements, the SCSP prototype keeps performing rather slowly for large databases. This means that it is still not suitable to be used in our architecture, where we need to replicate initial databases of 840.000 records (CSA size of 342 bytes) in a reasonable time. This task would take around 75 minutes with our prototype. The reason is that the protocol has a poor behavior for large database sizes. We conclude that this is mainly due to two different limitations. The first one regards the programming techniques used to implement the protocol (algorithms, data structures, etc) and was the main limitation in the former version. However, we consider that this factor has been minimized in the enhanced version with the improvements described in chapter 4. The second one is related to the SCSP itself. It has some design deficiencies that make it not suitable to operate with large databases. The most significant deficiencies can be summarized as follows:

- During the Cache Summarize phase, peers exchange summaries of their databases. Every time one of the servers goes down, this phase has to be performed again. Servers exchange messages filled with CSAS records. At any given time, both peers have at most one outstanding message. For large database sizes, this message exchange takes a long time. One possible solution would be the addition of a windowing system so that many messages could be sent at the same time.

- To acknowledge the reception of a CSA record, the recipient must send the entire CSAS record to the sender. This mechanism produces a huge amount of data to be transferred over the network. This could be solved by adding some kind of numbering to the CSA records, so that we could acknowledge ranges of records with the same message.

We must also point out that the SCSP is very similar to the OSPF protocol. Therefore, the results obtained in this document can be used to identify analogous deficiencies in OSPF. This could be useful in order to improve the performance in future versions of that protocol.

In order to address the lack of security in the architecture, we have implemented some security countermeasures, which are described in chapter 5. Among them, the most important is the SCSP security module, used to afford security to the SCSP traffic. We have chosen the Rijndael algorithm to implement the confidentiality service. The reason is that this algorithm is considered the most secure nowadays and it has been recently selected as the AES. Furthermore, it is the algorithm proposed to secure the future 3G networks. We have selected HMAC-SHA as the default authentication algorithm, as SHA has proven to be more secure than the obsolete MD5.

We have also performed some tests to show how the addition of these security services worsens the speed of the SCSP. As we expected, and despite the obvious overload due to the cryptographic computations, the use of the module does not dramatically affect the overall operation. In the worst scenario, where all SCSP messages are afforded both confidentiality and authentication/integrity, there is a 25% increment in the replication time. Even though we can tolerate this value, we could also reduce it by selecting which SCSP messages types we want to secure.

We have also presented an alternative SCSP security module based on IPSec. However, we have not been able to test the performance of this approach because there is no IPSec implementation available at the lab. We assume that the overload in this case would be very similar to the built-in solution providing the same security services and the same cryptographic algorithms are used. Nevertheless, we recommend the use of the built-in solution because it does not depend on whether our network supports IPSec and it provides a smart solution to the key distribution problem. Moreover, it allows us to select different security services for different message types.

this is not possible with the IPSec based module, which considers all the application traffic as a whole.

The auto-configuration module described in chapter 6 provides two basic services: intra-SG and inter-SG. The former one can be used to expand the existing SGs. The latter is aimed to facilitate the on-demand creation of new SGs. The basic version of the module deals with failures in a static way. Servers are connected by redundant links so that no reconfiguration is required when a failure occurs. The k-connectivity factor of the SG determines the maximum number of failures needed to disconnect the SG. The disadvantage of this approach is that it is not optimal in terms of traffic load.

To address this problem, we present an alternative auto-recovery procedure to be used in conjunction with the intra-SG and inter-SG services. This new approach minimizes the network load by using 1-connected SGs, which do not introduce any redundancy. The auto-recovery service is responsible for dynamically reconfiguring the SG topology when a failure occurs.

7.2 Future work

As we stated earlier, we consider that our SCSP prototype has almost reached the upper limit in terms of speed and performance optimization. Even though we could still improve some parts of the implementation, the replication times would not be significantly improved.

This leads to the conclusion that further work should focus on looking for new replication schemes for our architecture. We propose here two different approaches:

- Test other existing mechanisms. Many commercial and non-commercial database packages provide their own replication mechanisms, so that they are responsible for guaranteeing that remote databases have consistent information. They do not require an external synchronization protocol, since it is already included in the packages.

- Based on our SCSP experience, we can document all the SCSP limitations when large databases are being synchronized. This work could be used in the future to develop a new replication protocol, lighter than the SCSP, suitable for working with arbitrarily sized databases.

The IPSec based security module for SCSP could also be tested in order to establish a direct comparison between this proposal and the built-in module. Providing it proves to be an interesting approach in terms of performance, it requires further work to become operative. We just presented the basic proposal in this thesis, but there are still many issues to address, such as the key distribution mechanism. Furthermore, we would need to ensure

that this solution is fully compatible with the auto-configuration module proposed in chapter 6.

Finally, the future work regarding the auto-configuration module should focus on the auto-recovery procedure. While the intra-SG and inter-SG services have been fully tested, we only carried out very basic tests with that procedure. Furthermore, the procedure could be added new features to improve its functionality. For instance, we could use SCSP itself to replicate the SG topology information instead of using multicast messages for that purpose.

Appendix A

Cryptographic libraries

To implement the countermeasures proposed in this document, we have to develop some security functions that can be accessed by the existing SCSP implementation. Since the current code is written in C, we must use a cryptographic library available for the same programming language.

A.1 OpenSSL

OpenSSL 0.9.6b is a set of freely available cryptographic libraries that implements many of the features needed to build our solutions. The OpenSSL project is aimed to develop an Open Source implementation of SSL (Security Sockets Layer v2/v3) and TLS (Transport Layer Security v1) protocols, and it uses the SSLeay library developed by Eric A. Young and Tim J. Hudson to accomplish the required cryptographic tasks.

The library provides support for:

- Hashing functions: MD2, MD4, MD5, RIPEMD-160, SHA, SHA-1.
- Message Authentication Codes (HMAC).
- Symmetric cryptography: DES, IDEA, BLOWFISH, CAST, RC2, RC4, RC5.
- Public key cryptography: RSA, DH, DSA.
- X.509v3 certificate creation and handling.
- Creation of CRLs and CSRs.
- Data encoding: PEM format.

A.2 Cryptlib

Cryptlib 3.0 is a security toolkit aimed to provide an easy mechanism to add encryption and authentication services to applications. It provides implementations for the most popular symmetric and asymmetric algorithms, including the Rijndael cipher, recently selected for the AES standard. Since most of the algorithms are written in assembly language, the library offers a good performance and is suitable to work even for high-bandwidth applications such as video/audio and online network encryption.

A.3 Algorithm benchmarks

This section provides speed benchmarks for the cryptographic algorithms used in the implementation of the security module. All tests were performed in WS18. The main features of that machine can be found in table 4.2. Tables A.1 and A.2 show the obtained results.

Algorithm	**Library**	**Bytes processed**	**Time (seconds)**	**Megabytes(10^6 bytes)/second**
SHA-1	OpenSSL 0.9.6b	10^7	0,772	13,850
HMAC-SHA-1 (160 bits key)	Cryptlib 3.0 beta	10^7	0,922	10,846
Rijndael-128 (Encryption)	Cryptlib 3.0 beta	10^7	2,151	4,649
Rijndael-128 (Decryption)	Cryptlib 3.0 beta	10^7	2,314	4,321

Table A.1: HMAC-SHA-1 and Rijndael performance benchmarks

RSA-1149 (SHA-1) signature and verification performance Library used: OpenSSL 0.9.6b		
Data size (bytes)	**Signature time (milliseconds)**	**Verification time (milliseconds)**
100	71	5
1000	70	5
10000	71	5
100000	78	11

Table A.2: RSA performance benchmarks

Appendix B

Auto-configuration module SDL models

This appendix presents the SDL models developed to clarify the operation of the intra-SG and inter-SG auto-configuration procedures.

B.1 Intra-SG auto-configuration SDL model

This model considers a system with three different processes:

- **Unconfigured SCSP.** Represents an unconfigured SCSP server running the auto-configuration procedure.
- **Configured SCSP.** An existing SCSP server.
- **Application layer.** Represents the entity (protocol, application, etc) that is using the SCSP replication services.

A general view of the system including these processes and the set of signals used to communicate them is depicted in figure B.1. Figures B.2-B.7 show the SDL model, and a description of the signals is given in table B.1. The mechanism used to obtain digital certificates is out of the scope of the model and belongs to the system's environment. The interaction between the Application layer process and that mechanism is represented by the Query_Certificates and Obtain_Certificates signals.

B.2 Inter-SG auto-configuration SDL model

The SDL model for the inter-SG auto-configuration procedure is based on a system with three different processes:

- **Master Server.** Represents the server which receives the Create_SG signal.

- **SCSP Server.** An existing SCSP server.
- **Application layer.** Represents the entity (protocol, application, etc) that is using the SCSP replication services.

Figure B.8 depicts a general view of the system. Figures B.9-B.14 show the SDL model, and a description of the new signals is given in table B.2.

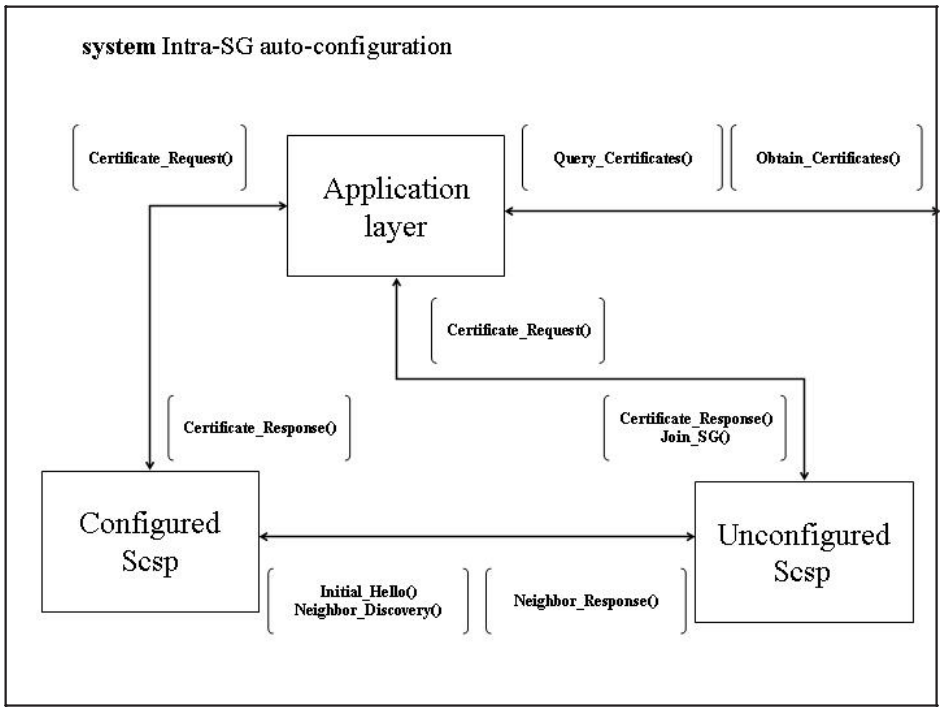

Figure B.1: Intra_SG auto-configuration system

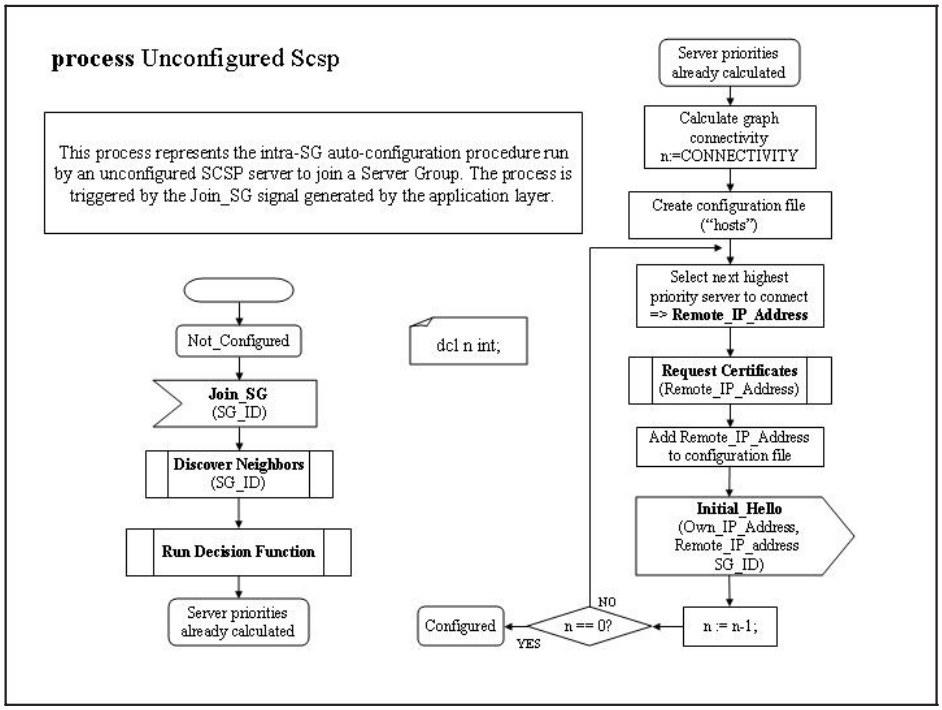

Figure B.2: Unconfigured SCSP process

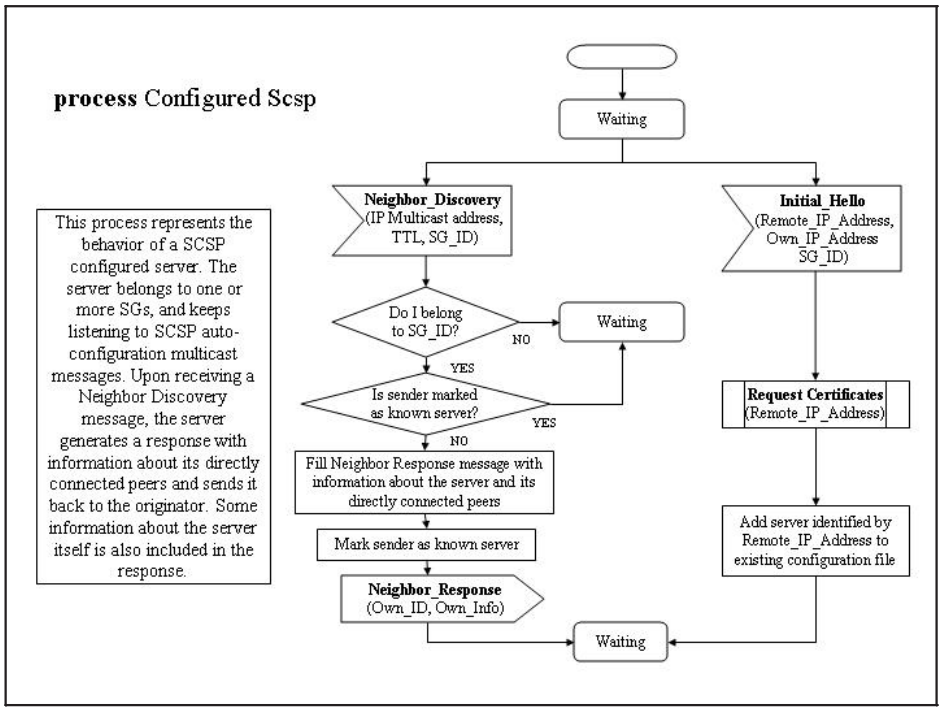

Figure B.3: Configured SCSP process

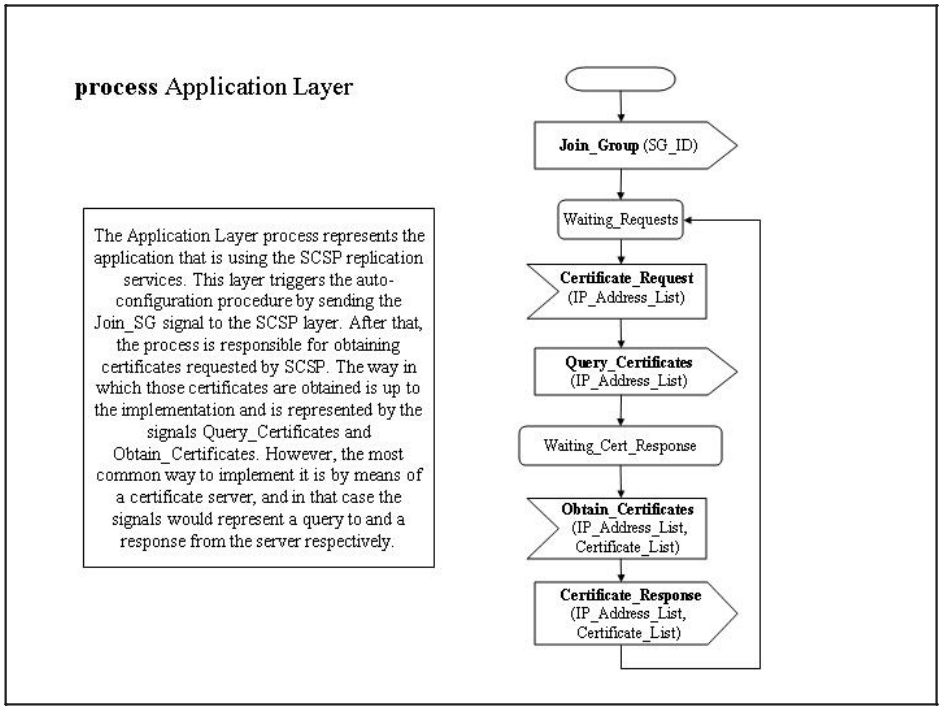

Figure B.4: Application Layer process (intra-SG)

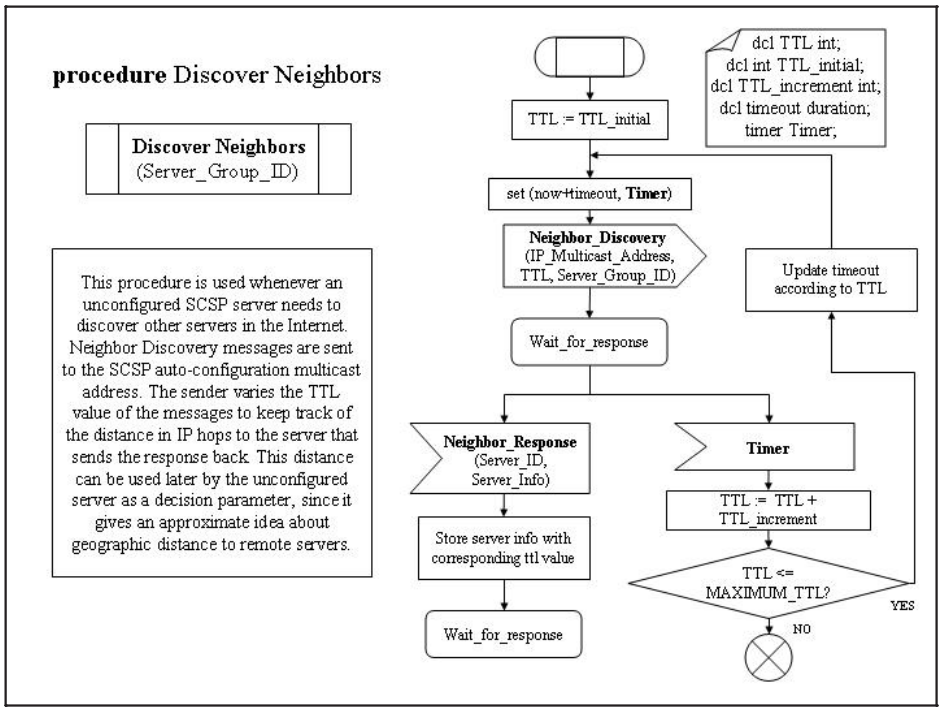

Figure B.5: Discover Neighbors procedure

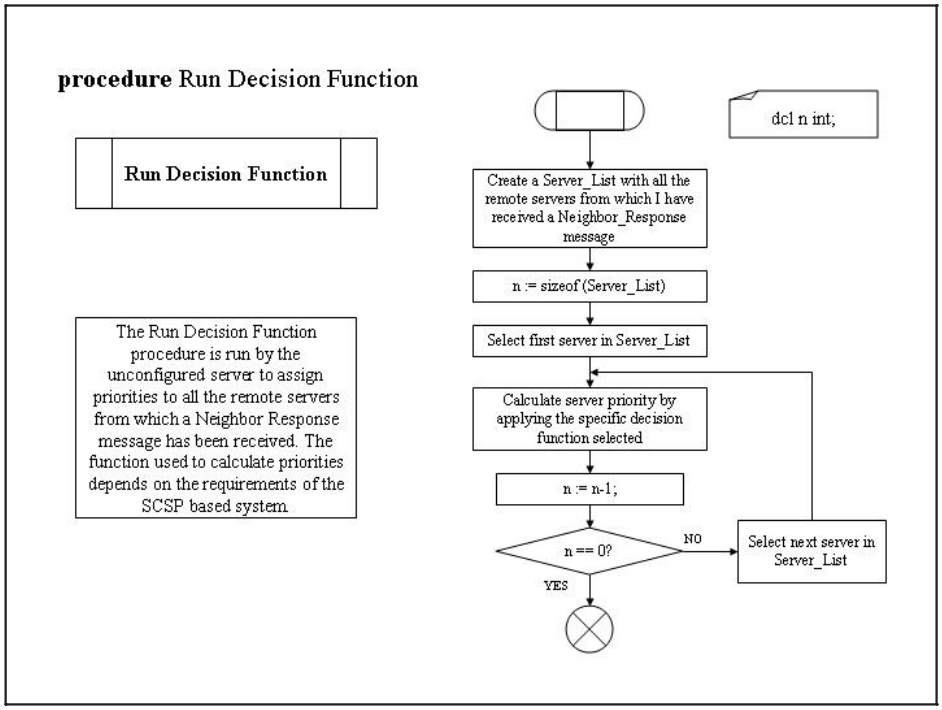

Figure B.6: Run Decision Function procedure (intra-SG)

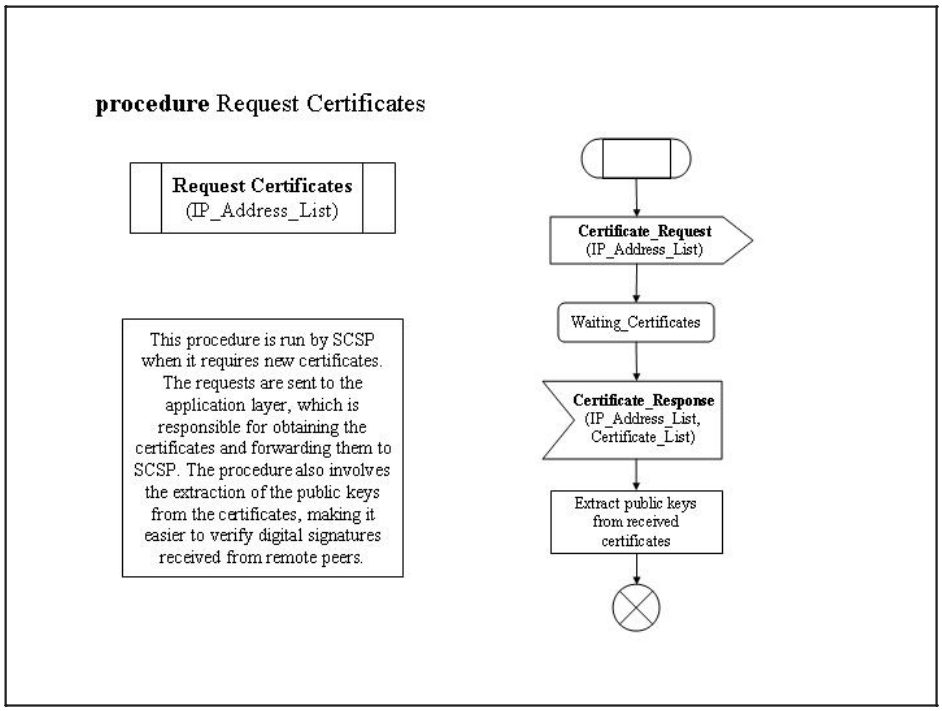

Figure B.7: Request Certificates procedure

Signal	**Description**
Join_SG (SG_ID)	Sent by the application layer to the control interface to trigger the intra-SG auto-configuration procedure. SG_ID identifies the SG the server has to join.
Neighbor_Discovery (IP_Multicast_Address, TTL, SG_ID)	Multicast message sent by the unconfigured server to discover remote servers belonging to SG_ID. IP multicast address is the SCSP auto-configuration assigned address. TTL is used to limit the scope of the message. The Neighbor Discovery message format is depicted in figure 6.3.
Neighbor_Response (Server_ID, Server_Info)	Message sent by Server_ID in response to a Neighbor_Discovery message. The message carries information about the server and its DCSs (Server_Info). The format of this message is shown in figure 6.3.
Initial_Hello (Sender_Address, Remote_Address, SG_ID)	SCSP Hello message sent by the unconfigured server to the remote selected peers. The recipient of this message detects that it comes from an unknown server and runs a procedure to add the server to its set of peers.
Certificate_Request (IP_Address_List)	Sent by SCSP to the application layer (using the control interface) to request the certificates of the servers identified by the IP addresses in IP_Address_List.
Query_Certificate (IP_Address_List)	Query sent by the application layer to the appropriate entity (usually a certificate server) to obtain the certificates corresponding to IP_Address_List.
Obtain_Certificates (IP_Address_List, Certificate_List)	Sent to the application layer in response to a Query_Certificate message. Certificate_List contains the requested digital certificates.
Certificate_Response (IP_Address_List, Certificate_Path_List)	Used by the application layer to inform SCSP about the file system paths where the certificates corresponding to IP_Address_List have been stored.

Table B.1: Signal description for the Intra-SG SDL model

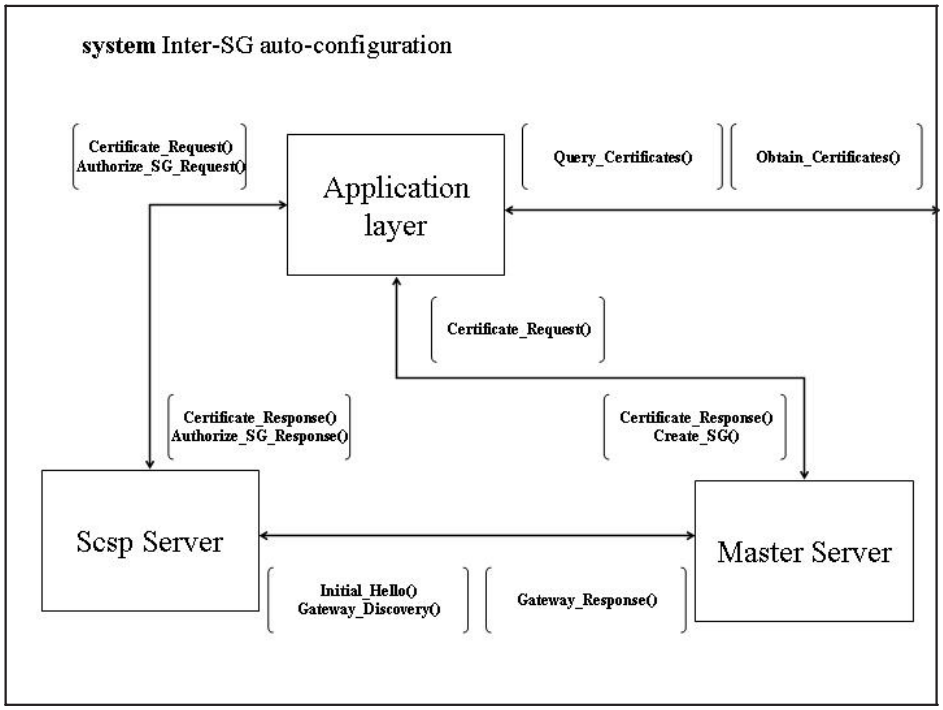

Figure B.8: Inter_SG auto-configuration system

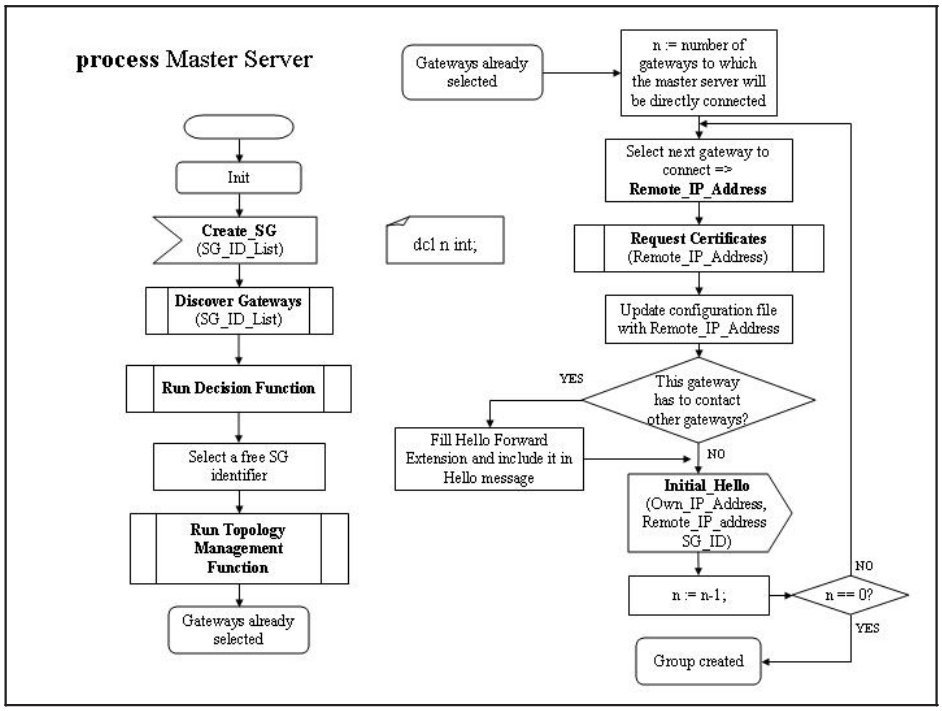

Figure B.9: Master Server process

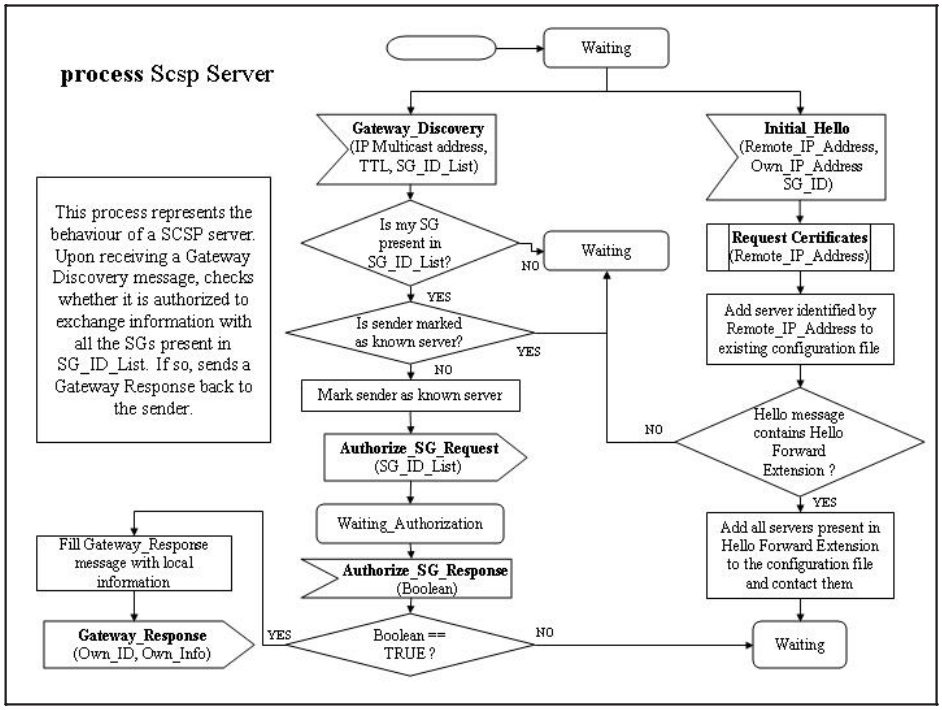

Figure B.10: SCSP Server process

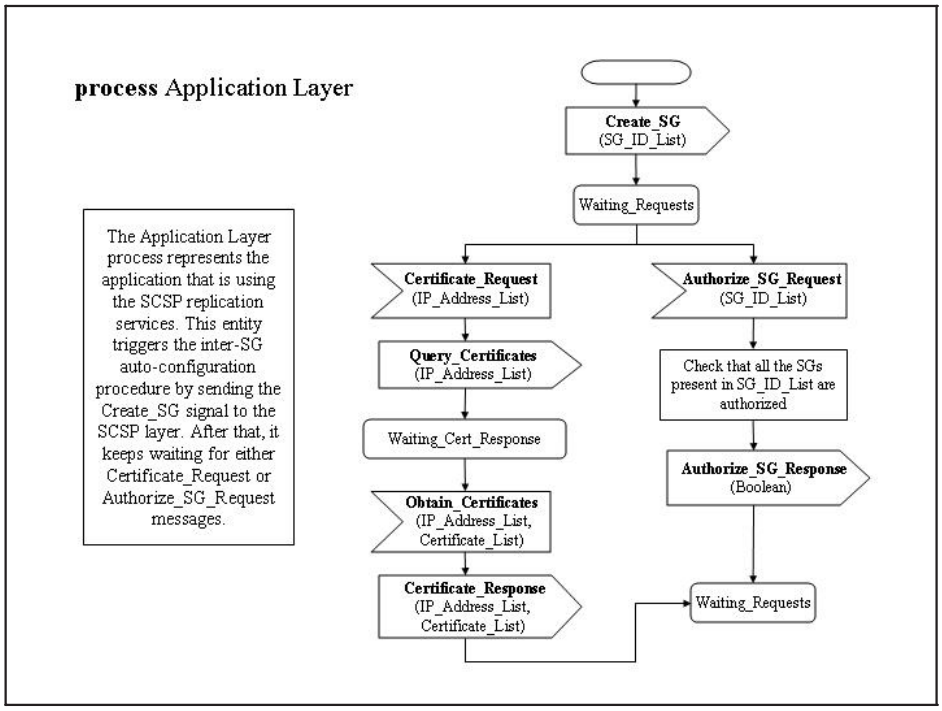

Figure B.11: Application Layer process (inter-SG)

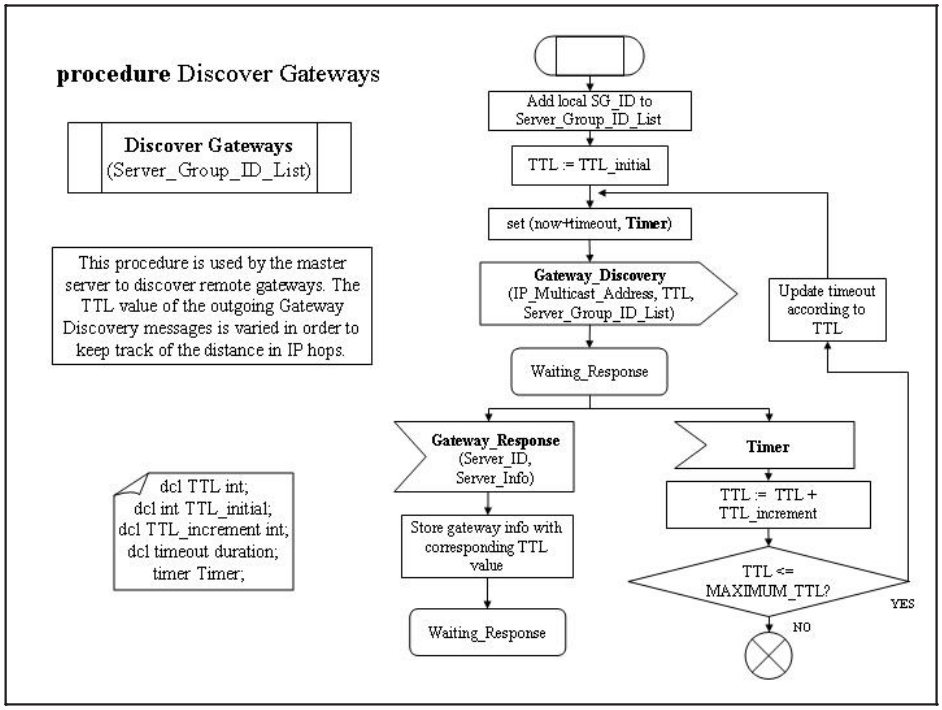

Figure B.12: Discover Gateways procedure

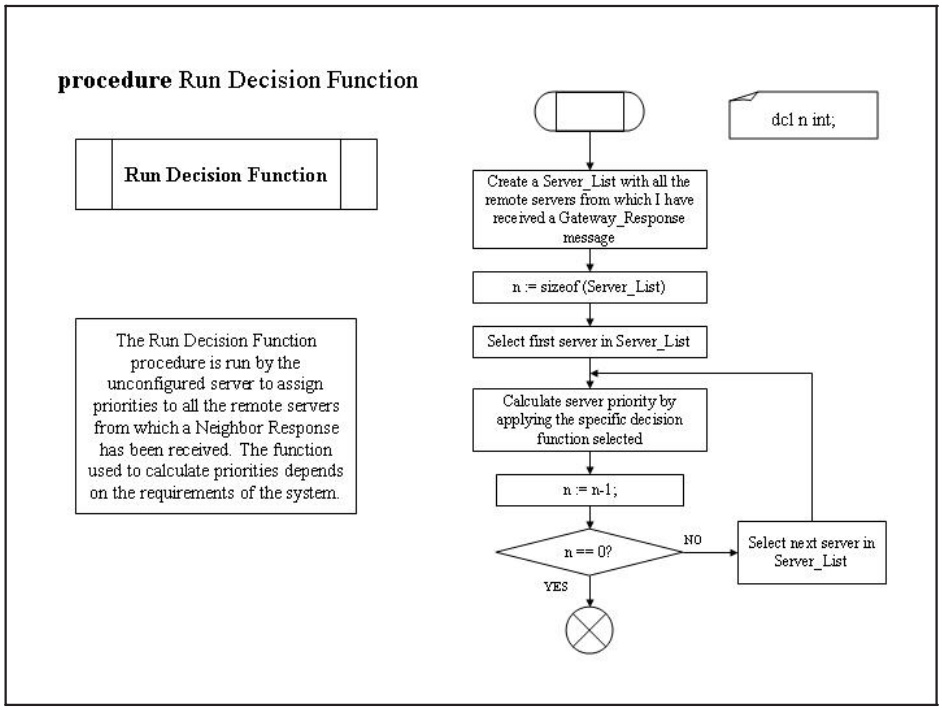

Figure B.13: Run Decision Function procedure (inter-SG)

Signal	**Description**
Create_SG (SG_ID_List)	Sent by the application layer to the control interface to trigger the inter-SG auto-configuration procedure. SG_ID_List identifies the SGs that will be involved in the new group.
Gateway_Discovery (IP_Multicast_Address, TTL, SG_ID_List)	Multicast message sent by the Master server to discover remote gateways. IP multicast address is the SCSP auto-configuration assigned address. TTL is used to limit the scope of the message. The Gateway Discovery message format is depicted in figure 6.4.
Gateway_Response (Server_ID, Server_Info)	Message sent by Server_ID in response to a Gateway_Discovery message. The message carries both information about the gateway and about the SGs it belongs to. The format of this message is shown in figure 6.4.
Authorize_SG_Request (SG_ID_List)	This message is sent by SCSP to the application layer to check whether the server is authorized to share information with all the SGs identified in SG_ID_List. This is needed to ensure that the gateway is allowed to join the new SG.
Authorize_SG_Response (Boolean)	Message sent by the application layer to SCSP in response to a Authorize_SG_Request message. The Boolean parameter tells SCSP whether the server is authorized to take part in the new SG or not.

Table B.2: Signal description for the Inter-SG SDL model

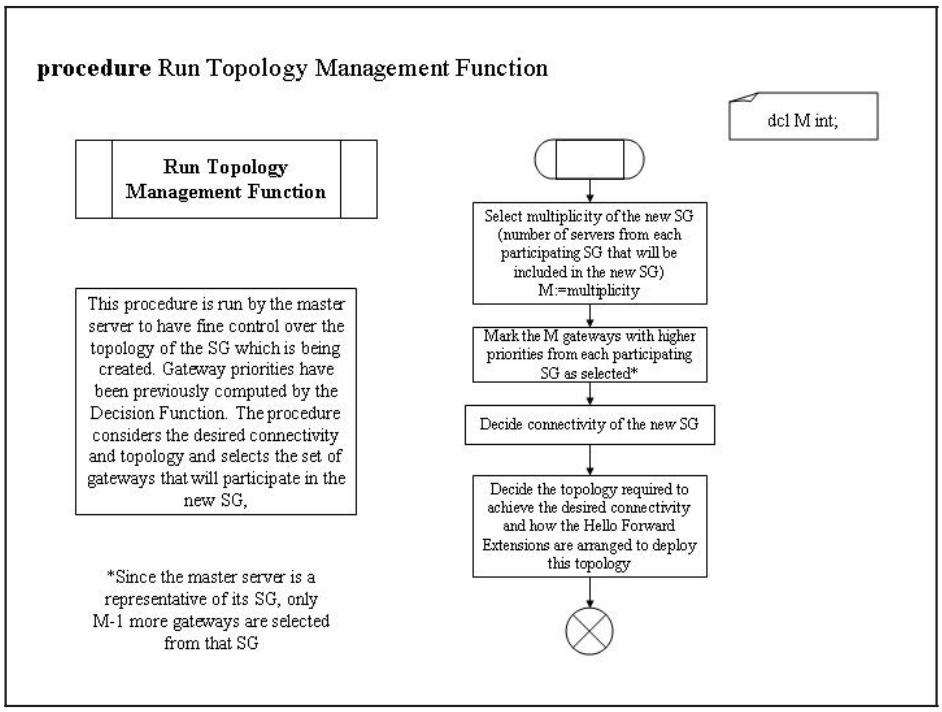

Figure B.14: Run Topology Management Function procedure

Bibliography

[1] Kantola, R., Costa Requena, J., Beijar, N. *Interoperable routing for IN and IP Telephony*. Laboratory of Telecommunications Technology, Helsinki University of Technology.

[2] Schulzrinne, H., Rosenberg, J. *The IETF Internet Telephony Architecture and Protocols*. URL: http://www.computer.org/internet/telephony/w3schrosen.htm

[3] Rosenberg, J., Salama, H. and Squire, M. *Telephony Routing over IP (TRIP)*. Internet draft, August 2001.

[4] Squire, M. *A gateway location protocol*. Internet draft, February 1999.

[5] Rosenberg, J., Schulzrinne, H. *A Framework for a Gateway Location Protocol*. Internet draft, 1999.

[6] Luciani, J., Armitage, G., Halpern, J., Doraswamy, N. *Server Cache Synchronization Protocol (SCSP)*. RFC 2334, April 1998.

[7] Rekhter, Y., Li, T. *A Border Gateway Protocol 4 (BGP-4)*. RFC 1771, March 1995.

[8] Moy, J. *OSPF Version 2*. RFC 2328, April 1998.

[9] Rosenberg, J., Schulzrinne, H. *A Framework for Telephony Routing over IP*. RFC 2871, June 2000.

[10] Chambers, C., Dolske, J., Iyer, J. *TCP/IP Security*. Department of Computer and Information Science, Ohio State University.

[11] Lucena, M. *Criptografa y seguridad en computadores*. Second edition, September 1999.

[12] Schneier, B. *Applied cryptography: protocols, algorithms, and source code in C*. Second edition, John Wiley & Sons, Inc. 1995, ISBN 0-471-12845-7, hardcover. ISBN 0-471-11709-9, softcover.

[13] Lenstra, A., Verheul, E. *Selecting Cryptographic Key Sizes*. November 1999.

[14] Kaliski, B. *The MD2 Message-Digest Algorithm*. RFC 1319, April 1992.

[15] Rivest, R. *The MD4 Message-Digest Algorithm*. RFC 1320, April 1992.

[16] Rivest, R. *The MD5 Message-Digest Algorithm*. RFC 1321, April 1992.

[17] SSH - Tech Corner. *Cryptographic algorithms*. URL: http://www.ssh.com/tech/crypto/algorithms.html

[18] Cryptography and Computer Security Lectures. Lecture 18. *Authentication, Hash Algorithms*. URL: http://www.cs.adfa.oz.au/teaching/studinfo/csc/lectures/less18.html.

[19] OpenSSL documentation. URL: http://www.openssl.org.

[20] The International PGP Home Page. *How PGP works*. URL: http://www.pgpi.org/doc/pgpintro.

[21] *Introduction to Public-Key Cryptography*. URL: http://developer.netscape.com/docs/manuals/security/pkin/contents.html.

[22] Housley, R., Ford, W., Polk, W., Solo, D. *Internet X.509 Public Key Infrastructure: Certificate and CRL Profile*. RFC 2459, January 1999.

[23] Kent, S., Atkinson, R. *Security Architecture for the Internet Protocol*. RFC 2401, November 1998.

[24] Harkins, D., Carrel, D. *The Internet Key Exchange (IKE)*. RFC 2409, November 1998.

[25] Kent, S., Atkinson, R. *IP Authentication Header*. RFC 2402, November 1998.

[26] Kent, S., Atkinson, R. *IP Encapsulating Security Payload (ESP)*. RFC 2406, November 1998.

[27] Krawczyk, H., Bellare, M., Canetti, R. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104, February 1997.

[28] Madson, C., Glenn, R. *The Use of HMAC-MD5 within ESP and AH*. RFC 2403, November 1998.

[29] Madson, C., Glenn, R. *The Use of HMAC-SHA-1 within ESP and AH*. RFC 2404, November 1998.

[30] Ramirez, J. *A Scalability Analysis of the Server Cache Synchronization Protocol (SCSP)*. Master's Thesis, May 2001.

[31] Costa-Requena, J., Kantola, R., Beijar, N., Ramirez, J., Gonzalez, I. *SCSP Scalability Analysis for Updating LNP Routing Information*. Networking Laboratory, Helsinki University of Technology, January 2002.

[32] Eastlake, D., Gudmundsson, O. *Storing Certificates in the Domain Name System (DNS)*. RFC 2538, March 1999.

[33] Boeyen, S., Howes, T., Richard, P. *Internet X.509 Public Key Infrastructure LDAPv2 Schema*. RFC 2587, June 1999.

[34] Rosenberg, J., Salama, H. *Authentication Attribute for TRIP*. Internet draft, June 2001.

[35] Costa-Requena, J. *An Implementation of the Server Cache Synchronization Protocol*. Master's Thesis, Helsinki University of Technology, Laboratory of Telecommunications Technology, 1999.

[36] Skiena, S. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Advanced Book Division, Addison-Wesley, Redwood City CA, June 1990. ISBN number 0-201-50943-1.

[37] A.V. Goldberg C code for the Maximum Flow problem. URL: http://elib.zib.de/pub/Packages/mathprog/maxflow/solver-1/index.html.

[38] Wang, C., Verrilli, C., Luciani, J. *Definitions of Managed Objects for Server Cache Synchronization Protocol Using SMIv2*. Internet draft, October 1998.

[39] SSH - Tech Corner. *Introduction to cryptography*. URL: http://www.ssh.com/tech/crypto/intro.html

[40] Kent, S., Lynn, C., Seo, K. *Secure Border Gateway Protocol (Secure-BGP)*. IEEE Journal on Selected Areas in Communications, Vol.18, No.4, April 2000, pp.582-592.

[41] Lynn, C. *X.509 Extensions for Authorization of IP addresses, AS numbers, and Routers within an AS*. Internet draft, October 1999.

[42] Branchaud, M. *A Survey of Public Key Infrastructures*. Master's Thesis, March 1997.

[43] Ellsberger, J., Hogrefe, D., Sarma, A. *SDL:Formal Object-oriented Language for Communicating Systems*. Prentice Hall, 1997.