IST IP CASCADAS "Component-ware for Autonomic, Situation-aware Communications, And Dynamically Adaptable Services" "

Bringing Autonomic Services to Life

WP4: Security, Survivability and Self-Preservation in Autonomic Communication Systems

Deliverable D4.1 – Security Architecture

Status and Version:	Final 1.2	
Date of issue:	19.12.2006	
Distribution:	Project Internal	
Author(s):	Name	Partner
	Ricardo Lent	ICL
	Pietro Michiardi	EURECOM
	Roberto Cascella	UNITN
	Anurag Garg	UNITN
	Ioannis Stavrakakis	NKUA
	Nikolas Laoutaris	NKUA
	Roberto Battiti	UNITN
	Alberto Montresor	UNITN
	Bruno Crispo	UNITN
	Laurence Hey	ICL
	Christos Xenakis	NKUA
	Georgios Loukas	ICL
	Gulay Oke	ICL
Checked by:	Roberto Cascella Peter H. Deussen	UNITN FOKUS

Executive Summary

This deliverable presents the CASCADAS Security Architecture and summarizes the work performed inside WP4 for the identification and the definition of new security principles and challenges typical of an autonomic communication system. Traditional security functionalities and more advanced security concepts are discussed and analyzed to show the natural evolution of the approach followed in designing the Security Architecture of the CASCADAS network infrastructure.

The envisioned security framework must at least guarantee protection from adversarial attacks typical of traditional networking systems that need not face problems arising from the heterogeneous capabilities of dynamic components such as those constituting the CASCADAS system. Therefore, we must deal with traditional security principles such as confidentiality, authentication, integrity and non-repudiation. Protection is guaranteed by using standard solutions based on cryptography algorithms that rely on key management and distribution services founded on symmetric or asymmetric keys. However, these schemes require either a-priori exchange of cryptographic material or the presence of a trusted third party that must negotiate keys for the communication.

CASCADAS cannot rely on a centralized entity or on static pre-defined trust relationships. The dynamic evolution of the network and the autonomic composition of services would suffer from static approaches. Moreover, the communication paradigm is shifted toward a service-centred network, with the ACEs (Autonomic Communication Elements) that participate in the formation of a service and at the same time make use of the service itself. In this context, we identify new security principles in autonomic systems: key distribution and agreement, secure information aggregation and cooperation.

The approach that we follow to secure component dynamic interaction considers the nature of ACEs. The CASCADAS system is formed by rational and irrational entities that are willing to either not cooperate actively toward the system goal or disrupt the network in the latter case. In this context we define the threats and the challenges that we have to face while securing the system and making it self-preserving. Moreover, we identify both attacks that mine the availability of the system, e.g. Denial of Service attacks, and attacks that are proper of the agent technology used to implement an ACE. Finally, we propose an analysis of soft attacks that derive from the selfish and malicious behaviour of some nodes.

In this context, we exploit the capabilities of the nodes to self-preserve and to sustain the network through self-management systems. We rely entirely on the nodes to evolve the network and to autonomically generate and provide services. As a consequence, the security architecture that we propose exploits the different functionalities of the nodes to provide security protection. According to the capabilities of the nodes and to their goal in the network we identify three different classes of security ACEs that have specific and elementary functions according to the capabilities of the ACE, so that we can enable component reusability and context adaptation of the ACE itself. The new security model proposed in this document is in line with the structure of the ACE and it is applicable to the interaction model proposed by WP1.

To show the applicability of the security framework and how components interact to secure a service provided within the CASCADAS conceptual model, we present the mock-up of the interactions for an application scenario. Furthermore, we give examples of how advanced security services evolve from basic security functionalities in the context of Denial of Service attacks. The conceptual deployment of the security architecture is aimed at supporting the self-preservation and -management of the system.

The way the security architecture has been designed paves the way for the next phase of the CASCADAS project where we will exploit the concepts and the solutions presented in this deliverable.

Table of contents

1 **Introduction** **6**
1.1 Purpose and Scope 6
1.2 Document History 6
1.3 Motivation of the deliverable 6
1.4 Structure of the document 6

2 **Problem Statement** **7**
2.1 Problem description 7
2.2 Assumptions 8

3 **Security principles** **8**
3.1 Traditional security issues 9
3.2 Basic security mechanisms 10
3.3 New security principles and mechanisms in autonomic systems 11
3.4 Presenting security in CASCADAS 14

4 **Entities and attacks in autonomic communication systems** **14**
4.1 Networks' evolution and generation of new challenges 15
4.2 Security threats & attacks (Adversaries) 16
4.3 Security requirements 17

5 **Defense measures: State of the Art and Self-Preservation mechanisms** **18**
5.1 Mobile Agents Technology and execution of services 19
5.1.1 Attacks on Mobile Agent Technology 19
5.1.2 Security framework: protection 20
5.1.2.1 Agent Platforms 20
5.1.2.2 Mobile agents 22
5.2 Access control 25
5.2.1 Trust Management 25
5.2.2 Trust Negotiation 27
5.3 Denial of Service 28
5.3.1 Defence 28
5.3.1.1 The Packet Source 28
5.3.1.2 Active tests 29
5.3.2 Detecting the existence of a DoS attack 30
5.3.2.1 Learning techniques 30
5.3.2.2 Statistical signal analysis 31
5.3.2.3 Multiple agents 31
5.3.3 Responding to an attack 32
5.3.4 Final remarks 33
5.4 Self-Preservation 33
5.4.1 A Sketch of the requirements for Self-Preservation 34
5.4.2 Self-Preservation by example 35
5.4.2.1 Distributed Selfish Replication 35
5.4.2.2 Distributed Selfish Caching 36
5.4.2.3 Peer to Peer Streaming 39
5.5 Soft-management to assure Self-preservation 41
5.5.1 Reputation-based trust management systems 41

5.5.1.1 Architectures of reputation management systems . . . 45
5.5.2 Social based trust management systems . . . 46

6 Roadmap **46**
6.1 Security in the framework of CASCADAS . . . 46
6.2 Autonomic approaches for self-preservation in CASCADAS . . . 47

7 CASCADAS Security Architecture **48**
7.1 Security capabilities of ACEs . . . 49
7.2 Components without security - ACE type A . . . 51
7.3 Cryptographic functions and services- ACE type B . . . 52
7.3.1 Goal Achievable . . . 53
7.3.2 Goal Needed . . . 54
7.3.3 Execution of security service . . . 54
7.4 Advanced security services - ACE type C . . . 54
7.4.1 Goal Achievable . . . 55
7.4.2 Goal Needed . . . 55
7.4.3 Security model for Self-Preservation: the case of DoS attack . . . 55
7.4.3.1 Defence Schemes: Basic Defence Scheme . . . 56
7.4.3.2 Defence Schemes: DDoS Protection based on Prioritisation and Throttling . . . 57

8 Mock up of ACE interactions **57**
8.1 Securing Inter-ACE Communication . . . 58
8.2 Securing Network Infrastructure . . . 59
8.3 Conclusions from the Mock-up . . . 60

9 Conclusion **60**

10 Bibliography **61**

Acronyms

Acronym	Meaning	Page
ACE	Autonomic Communication Element	8
PKI	Public Key Infrastructure	8
CA	Cetification Authority	10
TTP	Trusted third Party	10
KDC	Key Distribution Center	11
PGP	Pretty Good Privacy	15
p2p	peer-to-peer	15
MANET	Mobile Ad-hoc NETwork	16
DoS	Denial of Service	16
MAT	Mobile Agents Technology	19
PCC	Proof Carrying Code	21
PRAC	Partial Result Authentication Code	23
MAC	Message Authentication Code	23
CEF	Computing with Encrypting Function	23
PMI	Privilege Management Infrastructure	26
CRL	Certification Revocation List	27
TCP	Transport Control Protocol	28
IP	Internet Protocol	28
DDoS	Distributed Denial of Service	28
VoIP	Voice-over-IP	29
CAPTCHA	Completely Automated Public Turing Test to Tell Computers and Humans Apart	29
SPUNNID	Statistical Pre-Processor and Unsupervised Neural Net based Intrusion Detector	30
RBF-NN	Radial Basis Function Neural Network	30
TRA	Traffic Rate Analysis	30
ANFIS	Adaptive Neuro-Fuzzy Inference System	31
FCM	Fuzzy C-Means Clustering Algorithm	31
MARS	Multivariate Adaptive Regression Splines	31
LGP	Linear Genetic Program	31
SIM	Source IP address Monitoring	31
IAD	IP Address Database	31
BBA	Black Board Architecture	32
SOS	Secure Overlay Services	32
GT	Game Theory	33
MD	Mechanism Design	33
SO	Social Optimal	35
GL	Greedy Local	35
DSR	Distributed Selfish Replication	36
SC	Single Copy	38
MC	Multiple Copy	38
G_A	Goal Achievable	54
G_N	Goal Needed	54

1 Introduction

1.1 Purpose and Scope

This document presents the CASCADAS Security Architecture. It gives a first insight of the security requirements and the problems that might arise in the component interaction model envisioned for the CASCADAS project. It is supposed to provide the basic knowledge in terms of security measures and threats to other Work Packages, that should consider the results obtained as possible guideline to conduct their tasks.

1.2 Document History

Version	Date	Author	Comment
0.1	October 25, 2006	See front page	Initial contributions
0.2	November 24, 2006	See front page	Internal document
0.3	December 04, 2006	See front page	Added more sections
0.4	December 07, 2006	See front page	Integration of the sections
1.0	December 10, 2006	See front page	First version
1.1	December 28, 2006	See front page	Typos and corrections
1.2	January 19, 2006	See front page	Revised version following internal review

1.3 Motivation of the deliverable

This deliverable is a preliminary step toward the definition of the security framework suitable for the CASCADAS network. The work for the whole duration of the project has been divided in three different phases:

- provide an initial security architecture required by other Work Packages to guarantee their methods to function in a secure way if possible;
- exploration of new security concepts based on the feedback obtained within the context of the CASCADAS project;
- integrate the new concepts into the original framework to provide a complete security service when required.

The purpose of the deliverable is to detail the results accomplished by WP4. The document includes a general view of the security requirements for the CASCADAS nodes and provides solutions to the security problems that can arise while considering nodes’ autonomic interactions. Furthermore, the deliverable defines a security management framework to handle autonomic entities during their interactions.

1.4 Structure of the document

The document is organized to guide the reader toward the steps accomplished within Work Package 4 toward the definition and the deployment of the CASCADAS security architecture.

First, we state the problem that we address within WP4 and tackle in this deliverable. Then we clarify the assumptions that we have made for the design of the security architecture. These assumptions do not prevent a complete analysis of security threats and do not limit the applicability of the solutions presented (see Section 2).

Section 3 discuss the type of overlay network formed by the ACEs to communicate to exploit services and ACEs' capabilities. Based on the application type, two possible categories of autonomic overlay networks are defined: *managed* and *open* overlays. Since ACEs are fully autonomous and do not (or cannot) rely on any established infrastructure for the overlay initialization and operation, we focus on the latter. Thus, it is not possible to assume dedicated components to function as Trusted Third Parties. This Section also motivates the security concepts and principles traditionally considered in a distributed system. Furthemore, new security requirements and principles are defined within the framework of the autonomous service-centred network envisioned in CASCADAS.

The motivations and the new "identities" that might have part in the new paradigm of autonomic communication systems are clearly defined in Section 4, while the solution to address these issues have been surveyed in Section 5.

The state of the art solutions that we present exploit basic defence schemes such as identification and authorization of network entities to provide more complex security solutions. In this context we introduce the concept of two separated security models: hard or traditional security and soft security. Both compete to protect the network and are required to enable self-preservation of the CASCADAS system. In Section 5.1 we discuss threats and state of the art solutions applied specifically to Agent Technology that has many similarities with the ACE model. Then, we survey protection mechanisms to combat Denial of Service attacks in distributed environments (see Section 5.3).

We conclude Section 5 defining basic requirements for self-preservation in autonomic communication systems and we present examples and solutions derived from Game Theory and Mechanism Design to exemplify our ideas on self preservation in Section 5.4. Finally, we analyze existing self-management mechanisms such as reputation and trust based schemes to create a "soft web of trust" within a community of nodes in Section 5.5.

Section 6 defines security in the context of CASCADAS and outlines the approach we take in WP4 to handle security principles and requirements for an autonomic communication system.

The new concepts defined and discussed are applied for the definition of the security architecture which is outlined in Section 7. We define the roles of the security entities and their functions according to the capabilities of the nodes which are differentiated into three classes: A, B and C. The class of ACE states what are the actions an ACE can take in terms of security. ACEs belonging to class A do not have capabilities to use secure services and they are considered pssive elements. Security functionalities are demanded to class B and C with different purposes and security capabilitites. ACEs of type B implement hard or traditional security while those of type C have advanced security functionalities for the self-preservation and self-management of the system.

The verification of the ideas deployed is done by means of the mock-up presented in Section 8. An example is described and the message exchange and ACEs' interaction is clearly defined to demonstrate the feasibility of the designed architecture.

Section 9 concludes the document and discusses the interaction of WP4 with other WPs to integrate the security services deployed and presented within this deliverable to the concepts defined and studied in other research areas such as aggregation and knowledge network management.

2 Problem Statement

2.1 Problem description

The network envisioned for the CASCADAS project is based on the autonomic composition and aggregation of Autonomic Communication Elements (ACEs) to provide networking services in an adaptive and opportunistic manner.

Autonomic communication services are characterized by the involvement of heterogeneous entities without centralized organization or control, ranging from single users to entire institutions, to coalitions of institutions, and so on. These entities may actively contribute to the service definition/evaluation and self-management as well. Autonomic Communication Elements (ACEs) offer up their resources (content, access bandwidth, storage and CPU) and services for the benefit of other ACEs that are part of and at the same time use the system, in order to reach a common goal, i.e. provide an optimized value-added service. To deal with the complexity of real-world autonomic services, that involve large dynamic communities of users, the overall security framework should guarantee basic security functionalities to enable secure interactions and at the same time provide advanced security functionalities to enable soft-management and self-preservation of the entire system.

There are inherent security services required to create a world of trust to enable autonomic aggregation of ACEs and service provision such as authentication, message integrity and confidentiality, non-repudiation and identity management and trust relationship through digital signatures and credential management.

2.2 Assumptions

For the design of the security architecture of the CASCADAS system we assume that Autonomic Communication Elements (ACEs) have heterogeneous capabilities. Thus, cryptographic operations cannot be accomplished by all the nodes in the network. Furthermore, security mechanisms impact differently on network performances as additional resources are required to enable their execution. For the definition of the security architecture, the analysis of the bandwidth overhead given by the security mechanism is not considered.

We assume that there exists an infrastructure and routing algorithms to handle entities communication. ACEs are addressed by global identifiers and the message routing and network functionalities are out of the scope of the security architecture. Identifiers are required for identification and authorization purposes to ensure the appropriate use of available resources. We do not consider the definition of the identifier of the aggregated component and we handle a newly composed ACE as it would have multiple identifiers, e.g. those of the single components.

Since we are dealing with an *open* and autonomous system, we do not assume the presence of a predefined and single authority for authorization purposes and access control policies. Thus, there is no global Public Key Infrastructure (PKI).

We consider two classes of nodes that can strategize about their behaviour:

1. Rational nodes aim to maximize their expected utility from participation, given their beliefs about their environment, including the types of other nodes and the network topology. Rational nodes can exhibit behaviours normally attributed to the adversarial nodes found in cryptographic-protocol theory. Namely, rational nodes may use information learned from participation in a protocol to refine their own strategy. Rational nodes can also change or drop messages from other players, and can change or replace local algorithms as part of a utility maximizing strategy.

2. Irrational nodes behave strategically but do not follow a behaviour modelled by the mechanism designer. They behave irrationally with respect to the mechanism. For example, these nodes might have utility functions that depend on more than just their own preferences. "Antisocial" nodes, for instance, prefer strategies that hurt other nodes even when it means reducing their own economic utility.

3 Security principles

An autonomic communication system can be viewed as an overlay network formed by a collection of dynamic entities (that we call ACE, *i.e.* Autonomic Communication Entity) without the aid of a dedicated infrastructure used to carry out basic communication primitives such as unicast and broadcast nor to setup and manage security associations among them.

Different kind of services and applications can be deployed on and at the same time use the overlay network formed by the ACEs to communicate, share information (both application level and system state data) or execute specific or complex tasks. Based on the application type, we envision two possible categories of autonomic overlay networks: *managed* and *open* overlays. We refer to *managed* overlays when bootstrapping the logical connectivity can be supported by a temporary centralized infrastructure and when an *a-priori trust* between the ACEs that form the overlay is available. A-priori trust relations can be established, for example, by a single authority managing the ACEs or by a common organizational structure of the end-users or groups that deployed the ACEs.

On the other hand, in an *open* overlay network, ACEs are fully autonomous and do not (or cannot) rely on any established infrastructure for the overlay initialization and operation. Furthermore, since ACEs are operated by independent end-users or groups that do not necessarily belong to the same organization nor share a single authority, an *a-priori trust* relation between them is not available. Trust between nodes has to be built through specific mechanisms tailored to the challenging scenario offered by an open environment.

The claim exposed in this Deliverable is that overlay network operation and use can be easily jeopardized if countermeasures are not embedded into basic communication primitives at the early stages of their design. Security is essential because unlike networks using dedicated components to function (*i.e.*, Trusted Third Parties (TTP), Routers administered by a Trusted Authority), in an autonomic network those components are operated by all available entities that compose the system. This very difference is at the core of the security problems that are specific to autonomic systems.

3.1 Traditional security issues

Traditionally, security has been reckoned an important issue for infrastructure networks (i.e. networks in which dedicated components, such as routers, provide the basic network operation), especially for those running security-sensitive applications. To secure the network infrastructure and the applications designed for such type of dedicated networks, the following basic security services have been widely addressed by the research community: confidentiality, integrity, authentication, and non-repudiation.

- *Confidentiality* ensures that certain information is never disclosed to unauthorized entities. Network transmission of sensitive information, such as strategic or economically valuable information, requires confidentiality. Leakage of such information to an eavesdropper could have severe consequences.

- *Integrity* guarantees that a message being transferred is never corrupted. A message could be corrupted because of benign failures or because of malicious attacks on the network.

- *Authentication* enables a node to ensure the identity of the peer node it is communicating with. Without authentication, an adversary could masquerade a node, thus gaining unauthorized access to resource and sensitive information and interfering with the correct operation of other nodes.

- *Non-repudiation* ensures that the origin of a message cannot deny having sent the message. There are other security goals (e.g., authorization, intrusion detection, etc...) that are of concern to certain applications, that we will discuss later in this section.

In the following, we present an overview of basic security principles for autonomic networks that have their analogue in infrastructure networks and discuss on the basic mechanisms that enable their provision. We then present new security exposures that are specific to the autonomic paradigm and briefly introduce the security requirements raised by such additional threats.

3.2 Basic security mechanisms

Basic security services such as confidentiality, integrity, authentication, and non-repudiation have been the subject of intense research in the last thirty years. It is out of the scope of this section to provide a survey on the various types of cryptographic algorithms that have been proposed in the literature. Instead, we discuss on the salient aspects of the two main families of cryptographic algorithms that represent the key-enabler to deploy traditional security services in infrastructure networks.

The choice of keying material that is used to provide network security services is of fundamental importance. The cryptographic algorithms used by basic security mechanisms all rely on a *key management and distribution* service and can be based on **symmetric** or **asymmetric** keys. Depending on the key type, algorithms based on symmetric keys are called *symmetric algorithms* while algorithms based on asymmetric keys are called *asymmetric algorithms* or public key algorithms.

The choice of the algorithm type has an immediate impact on the requirements in terms of *computational power* available at each node of the network. It is known from the literature that computational power requirements are significantly higher for asymmetric algorithms. Moreover, the effects of the algorithm type can be extended to take into account the *energetic cost* related to the execution of the security protocol. Furthermore, depending on the algorithm type, a security protocol can introduce a non negligible *traffic overhead* in the network: the distribution and the utilization of keying material have an impact on the length of the messages exchanged by the nodes thus leading to a significant waste of bandwidth resources.

For traditional centralized systems, security mechanisms based on asymmetric keying material have received a lot of attention. The RSA cryptosystem is a seminal work that allowed the growth of modern asymmetric algorithms. RSA [97], is most commonly used for providing confidentiality and ensuring authenticity of digital data. These days, RSA is deployed in many commercial systems. It is used by Web servers and browsers to secure Web traffic, it is used to ensure confidentiality and authenticity of e-mail, it is used to secure remote login sessions, and it is at the heart of electronic credit card payment systems. In short, RSA is frequently used in applications where security of digital data is a concern. Traditional public key cryptosystems are based on the RSA cryptosystem and require any user’s public key to be certified with a certificate issued and digitally signed by a Certification Authority (CA): this is done in order to provide authenticity of the public key belonging to a user. Any participant who wants to use a public key must first verify the corresponding certificate to check the validity of the public key. When many CAs are involved between two users, trust relationships between those CAs also need to be verified. In fact, public key cryptosystems rely on a public key infrastructure (PKI) that is used in order to manage the trust relationship between entities (eventually in a hierarchical manner). However, traditional certificate-based schemes are affected by the main drawback that is key revocation: revocation of public keys and the corresponding certificate is a big issue and requires a large amount of storage and computing. Moreover, key revocation services often require some trusted authority to be on-line.

Some of these limitations have been addressed and partially mitigated in the literature with the introduction of Identity Based Cryptography. The idea of identity based cryptosystems was proposed by Shamir [105] with the original motivation of simplifying certificate management in email systems, thus avoiding the high cost of the public-key management and signature authentication in cryptosystems relying on a public key infrastructure.

ID-based systems *avoid the explicit authentication of public keys* using public-key certificates by offering a cryptosystem wherein *the identity of a user plays the role of his public key*: each entity’s public key can be defined by an *arbitrary string*, *i.e.* users may use some well-known information such as email addresses, IP addresses or any other unique identifier as their public key. From a user’s identity (which is publicly known and in a standardized form), a TTP (e.g. a key distribution center) computes the corresponding private key and securely transmits it to the user.

The original goal of Shamir was only partially achieved by a few solutions until Boneh-Franklin [10] proposed the first practical identity based encryption scheme based on Weil Pairings on elliptic curves. Since then, several other identity based cryptography schemes [20, 41, 16, 86, 23] have been proposed. The common denominator of ID-based cryptosystems available in the literature is that they provide the same services offered by a PKI without the management costs of a PKI: by contacting a key distribution center (KDC), a user receives a secret key corresponding to the public key derived from the user's identity.

The main issue of such systems is that the KDC possesses all secret keys corresponding to the users of the system, i.e. all proposed schemes have inherent key-escrow properties, that is an arrangement exists stating that the keys needed to decrypt encrypted data are held in escrow by a third party, so that someone else (eventually a malicious entity) can obtain them to decrypt messages which they suspect to be relevant.

This limitation has been addressed, for example, in [23].

The security principles and algorithms illustrated so far constitute the basic building blocks to provide communication and information security. However, the key enabler component behind the techniques used to authenticate entities and ensure message confidentiality and integrity is represented by key management and distribution.

What are the novel issues that characterize a situated and autonomic system and that needs to be addressed in order to provide security services from the simple cryptographic primitive to more complex ones? We try to address these specific issues in the following, with the goal of emphasizing the new principles required by an autonomic system.

3.3 New security principles and mechanisms in autonomic systems

In this section we overview what we believe to be the novel requirements raised by autonomic systems. We briefly discuss on why recent techniques available in the literature are not adequate and provide new directions that we believe to be capable of addressing the specific needs of an autonomic system.

Key distribution and agreement

Key management and distribution can be viewed as the basic building block necessary to provide cryptographic service from the basic to the more complex ones.

For example, in symmetric key cryptography both parties must possess a secret key which they must exchange prior to using any encryption. Distribution of secret keys has been problematic until recently, because it involved face-to-face meeting, use of a trusted courier, or sending the key through an existing encryption channel. The first two are often impractical and always unsafe, while the third depends on the security of a previous key exchange.

To address these problems, in public key cryptography the distribution of public keys is done through trusted servers. When an entity creates a key-pair, she keeps one key private and the other, public-key, is uploaded to a server where it can be accessed by anyone to send the user a private, encrypted, message. The private keys are never transmitted, and can therefore be physically secured.

In Section 3.2 we briefly introduced two fundamental cryptosystems that rely on a centralized and trusted entity to distribute keying material (and to manage it). However, these systems cannot be readily used to secure autonomic systems since they all involve a centralized component used to establish the necessary security associations for the system to be secure. In the literature it is possible to find recent efforts toward the distribution of this centralized role to all the system participants. Techniques akin to *polynomial secret sharing* [68], wherein a technique based on Lagrange interpolation is used to enable distributing a secret amongst a group of participants each of which allocated with a share of the secret, have the potential of distributing the role of a centralized authority to sign keying material used in later communications. However, these techniques are not adequate for a dynamic environment wherein new entities can suddenly join and overload the system or existing ones leave without prior notification.

To surmount these problems, other mechanisms that rely on the concept of *Web of Trust* have been proposed in the literature of self-organizing systems [11]. In these systems, trust relationships are transitive and refer to local repositories maintained by every entity in the system and containing the public keys of other trusted entities. Although appealing, these solutions are not in line with the authentication requirements raised by autonomic systems. Indeed, anyone can generate a number of key pairs and authentication is not guaranteed. Moreover, in case authentication is not required, these systems have their foundations in public-key cryptosystems and are thus computationally demanding.

In some specific cases, such as the one offered by sensor networks, it is possible to envision the *pre-distribution of keying material* [132] before the deployment of the network. Then, stochastic algorithms are used to establish cryptographically secure path from a source to the destination of a message. Key pre-distribution mitigates the problem of bootstrapping security associations among untrusted entities, but fail in a dynamic scenario.

Autonomic systems call for key distribution and agreement mechanisms that are able to support a scalable and dynamic system, that are distributed in nature, and that do not rely on any pre-established trust relationships. We believe that techniques that use the epidemic nature of the communication in such systems such as *key infection* [5] are the most suitable to address the inadequacies of previous proposals. As opposed to prior works, we will not assume the existence of a global and ubiquitous attacker nor use *strong* attacker models. Instead, we will target a *relaxed attacker model* wherein the attacks are limited to passive attacks and cannot cover a large portion of the network. This assumption can be understood by means of a simple example: consider a scenario whereby a sensor network is deployed to provide measuring services. Traditional security assume an attacker capable of disrupting the entire system by tapping into all possible communication links, for example with the goal of acquiring sensed data. In our attacker model, we assume malicious entities capable of compromising only a *feasible* portion of the network: in other words, we model attackers in a more realistic way as it would be quite difficult and surely very expensive to devise a device capable of tapping the communication of a network deployed on several thousands of square meters.

Secure information aggregation

The autonomic nature of the system addressed by the CASCADAS project calls for the continuous monitoring of the activity of the system for the purpose of self-preservation and self-healing.

In CASCADAS, a pervasive and distributed supervision component measures state information of the overlay network components, the ACEs, and diffuse this information throughout the system so that single components can use it and react to a critical system state. This process can be enriched further by allowing ACEs to *gossip* their **local** state information and incrementally compute **global** state information based only on simple messages diffused in the system in a stochastic way. The diffusion of both unaltered and aggregated information can be characterized through models akin to the epidemic diffusion of a virus or disease in a biological system.

Although information aggregation, or in general in-network processing, can be a very effective mean of reacting to systems' dynamics, it is easy to understand the impact of an attack directed at this process. We term the vulnerability of the monitoring component and the basic communication mechanisms used to diffuse information as *poisoning attacks*. Poisoning attacks exploit the diffusion speed of the underlying communication primitive used to propagate information in the overlay network and can have the dramatic impact of rendering the system instable, less reactive and insecure. The need for security mechanisms for this new type of attacks is stringent as the very nature of the CASCADAS system relies on a feedback loop containing state information from the system's components that must be secured.

The intuition of applying traditional cryptographic techniques such as encryption or digital signatures to secure sensitive information only partially solves the problem. Due to the computational requirements raised by intensive verification of messages exchanged by ACEs it is not reasonable to adopt a *hop-by-hop* message verification, *i.e.* the decryption and re-encryption of sensitive data performed at every ACE of the system. Instead, we propose to adopt special *homomorphic cryptographic constructs* that avoid unnecessary cryptographic operations while at the same time allow computations over encrypted data. As a simple example, an homomorphic cryptosystem is a mechanism for which the following holds:

$$E[a] + E[b] = E[a + b]$$

where $E[]$ stands for the encryption operation.

Cooperation issues

The autonomic paradigm addressed by the CASCADAS project targets networks that can be viewed as distributed systems composed of autonomous entities, which need *cooperation* in order to work properly.

The basic requirement for making the cooperative paradigm operational is the supposed contribution of all entities that compose and, at the same time, make use of the system. However, the deployment of overlay networks composed of independent ACEs relaxes the assumption on nodes cooperation because ACEs cannot be assumed to be coordinated for pursuing the same goal. This implies that as long as applications of autonomic network envision an objective common and profitable to all ACEs, cooperation among nodes can be assumed. In the context of civilian or commercial applications, the ACEs typically do not belong to a single authority, and consequently their cooperation cannot directly be assumed. The lack of any centralized authority, guaranteeing the overall collaboration, motivates a possible tendency of entities to misbehave by not adhering to the cooperative paradigm.

Generally, an entity that does not cooperate is called misbehaving. Such misbehaviour can be caused by entities that are *malicious* or *self-interested*. Malicious entities aim at breaking the proper system operation to intentionally damage others. This kind of misbehaviour rise several security problems that can be countered through traditional security services, such as the ones illustrated in the previous Section. As an example, a malicious entity can perform an overlay routing disruption attack whereby the attacker sends forged routing packets to create routing loops or to partition the overlay network. This kind of attack can be prevented through the integrity and authenticity verification of overlay routing control messages.

On the other hand, a self-interested entity does not intend to directly damage the overall system functioning, but is unwilling to spend its resources (bandwidth, storage, energy, *etc ...*) on behalf of others. From this kind of misbehaviour a *new class of problems*, that we group in non-cooperation problems, arises. Lack of cooperation has gained much relevance in the context of self-organizing systems and several techniques akin to game theory and mechanism design have been proposed to counter this new type of threat. We claim that analytical tools such as game theory are of fundamental importance to understand and predict the system's state when self-interested components interact while pursuing their own individual goals.

A special case that characterizes the CASCADAS vision of an autonomic system is represented by self-aggregation. The work carried out in WP4 in the context of cooperation is strongly related to WP3 [21], where the goal is to investigate the global self-aggregation dynamics arising from local decision-based rewiring of an overlay network, used as an abstraction for an autonomic service-oriented architecture. Self-aggregation allows ACEs to establish and modify logical neighbourhood relationships (that is, the overlay network is dynamic in nature) in order to adapt to the current context of the overlay network. The situated system envisioned by CASCADAS makes use of the distributed monitoring component to gather system state information such as *load* and *congestion* to make two representative examples, and modifies accordingly the overlay network that connects the different ACEs [2]. What is the impact of a misbehaviour, be it due to a malicious activity or to self-interest, on the aggregation process? What are the possible countermeasures? These issues can be addressed by taking a close look at the very nature of the problem. In a similar way as for the need for cooperation to correctly operate the overlay network, the case of self-aggregation calls for a modelling tool that allows to analyze the conflicting interests that drive ACEs during the reorganization of the overlay network. Besides the evaluation of the impact of a malicious activity on the system, that can be mitigated through traditional security countermeasures, we claim that the theoretical foundation offered by game theory is necessary to analyze these issues and devise distributed incentive schemes to guide the system behaviour to a convenient state.

3.4 Presenting security in CASCADAS

The work carried out in WP4 aims at identifying a wide area of security related problems that affect those systems targeted by the CASCADAS project.

In Section 4 more details are provided to explain how the network, following the CASCADAS vision, evolves and what are the specific challenges that are proper of a dynamic and distributed environment. Specifically, a lot of emphasis is devoted to discern what has been traditionally looked at as *hard security* problems, whereby a system is whether under attack or safe giving thus a classical boolean vision of security, from a new form of system security that allows the existence of grey zones in which a system may be under attack, but still able to recover from an abnormal system state. As a first step to identify those new security challenges, Section 4 defines the sources of attacks in such distributed system by describing categories of entities that are differently involved in the participation to the network. Security requirements are also discussed as a complement and guiding rule that can also be seen in the literature.

The solutions implemented in the literature to reduce the risks of attacks are presented in detail in Section 5. Section 5 is devoted to survey traditional security mechanisms and countermeasures that have applied to secure network operations. Ranging from basic defense schemes to more complex ones, the concepts introduced in Section 3 and Section 4 are used to describe how self-preservation can be achieved in the CASCADAS system. Techniques ranging from Mobile Agent Technology to Access Control Mechanisms that have been applied to protect the network from unauthorized use of resources are deeply analyzed in Section 5. Finally, a lot of emphasis is given to trust and reputation management systems as they play a key role to enable the creation of trust among individuals (such as ACEs). Mechanisms derived from traditional Game theory and Mechanism Design theories to deal with rational nodes are also surveyed in detail.

4 Entities and attacks in autonomic communication systems

The previous section has outlined basic security principles and proposed new principles that are proper of an autonomic communication system. The deployment of solutions that solve security breaches in distributed and autonomic systems are still not completed defined as the nature of the attacks and the nature of the network pose several constraints on the applicability of traditional security.

Furthermore, autonomic systems such as CASCADAS are formed by rational and irrational entities that are willing to either disrupt the network or to not cooperate actively toward the system goal. In this section we discuss in more detail how the network has evolved and what are the specific challenges that are proper of a dynamic and distributed environment. Then, we define the sources of attacks by defining two categories of entities which have different scope for their participation to the network.

The solutions implemented in the literature to reduce the risks of attacks are presented in detail in Section 5.

4.1 Networks' evolution and generation of new challenges

The first forms of network applications adopted the client-server model, where a client program, which runs on one host, requests/receives services from a server program, which runs on another host. In such system, where every node has a clearly defined role, the issue of network security can be satisfied by using traditional security mechanisms, also called as *hard security* [94]. The purpose of these mechanisms is to protect the resources from malicious users, allowing access only from authorized users. Examples of such security mechanisms are *authentication* and *access control*. Authentication provides the so called *identity trust* [50], which describes the belief for the correctness of an agent's identity. Identity trust is a condition for trusting a party behind an identity and is implemented by typical authentication schemes such as X.509 and PGP [133]. The verification and managing of identities is performed by authentication service providers, while access control is provided by access trust systems [37]. These systems ensure *access trust* [50], which describes trust in principals for accessing resources owned by or under the responsibility of a relying party.

In recent years, there has been a growth of e-commerce and peer-to peer systems, which creates new security challenges that are not met by traditional approaches. Peer-to-peer (p2p) networks are self organized, distributed systems, which use the unexploited resources of networks. These networks consist of equivalent nodes, where each of them has the ability to implement the function of both, client and server. This means that a peer application in a user's host system operates both as client and server, and, thus, p2p nodes are also called servent. Therefore, because of the cooperation between end-applications, the amount of distributed resources that can be collected is very large, and this has led the rapid growth of users' interest for p2p applications.

The first generation of p2p applications referred mainly to file sharing applications (multimedia file exchange, e.g., audio, video), while the next generation it will be based on the concept of cooperative distributed resource sharing. The resources that can be distributed are information, storage capacity, processing power and bandwidth. An example of distributed computational resources is found in Grid[1] architectures. In such environment, where information providers can act deceitfully by providing false or misleading information, traditional security mechanisms are unable to protect from this kind of threats. This happens because the security mechanisms should protect from those who offer resources, and this problem is in fact the reverse of what the traditional security mechanisms have to deal with (i.e., the traditional security mechanisms mainly protect from those who try to steal resources. This security approach is described by the term *soft security* [94], which is also called *social control mechanism*. Examples of such mechanisms, which protect against the aforementioned security threats, are *trust and reputation systems*.

Trust and reputation systems are useful tools for deriving *provision trust* [50], protecting in that way the relying parties from malicious or unreliable service providers. Provision trust is users' knowledge about the reliability of authenticated parties, or the quality of goods and services that they provide. In other words, provision trust describes the relying party's trust in a service or resource provider. Due to the creation and maintenance of provision trust in a system, participants rely on "future payment", which guarantees (creates to the participants a sense of security) that the investment the participants have done when they shared their resources, will be returned to them. Therefore, the participants are willing to contribute their own resources, because they feel sure that this effort (and attitude) will be recognized and rewarded. This means that in the future they will also receive good behaviour from other participants in the system or they will enjoy the privileges being offered by the incentives mechanisms, which are incorporated in the reputation system. Conclusively, the creation and maintenance of provision trust in a system is necessary in order to function properly, that is to converge to the desired (ideal) collaborate behaviour between the participants. Without provision trust, converge to that behaviour would not be possible.

[1]"An infrastructure that enables flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions and resources" Ian Foster and Carl Kesselman

It should be mentioned that the application of soft security mechanisms is not a replacement of traditional security solutions (hard security), but it is a complementary strategy that works through establishing trust between participants. An example of this supplementary coexistence is found in p2p networks. The success of these networks is because of their distributed characteristic, as we mentioned before, and their deployment features, such as anonymity. A system that offers anonymity is necessary to incorporate a mechanism that offers identities (pseudonyms), so that it can distinguish the users between each other and match the pseudonym with the owner's behaviour (i.e. with the behaviour history). In addition, this mechanism should provide identity trust, not allow forgery, and must be spoof-resistant. Finally, we have to mention that the effectiveness of security measures (i.e. unforgeable and spoof-resistant) is a function of cost (CPU cycle, bandwidth, money).

4.2 Security threats & attacks (Adversaries)

Each part in an autonomous system is exposed to adversaries' attacks. The term "adversary" means an agent whose intention is to harm other peers or the whole system, or it seeks to act in a manner that is in contrast with what is considered to be an acceptable behaviour in the network. Based on the objectives that an adversary has in a system, he is ranked in one of the following two categories.

The first category of adversaries includes the free-riders also called as selfish or rational. These entities consume system resources or services, while they contribute little or none of their resources. Their scope is to maximize their personal benefit at the expense of other participants in the system. An example can be found in a Mobile Ad-hoc NETwork (MANET), where free-riders nodes inject their own packets in the network, but they do not participate as intermediate nodes in the network, meaning that they do not forward any packets from other nodes. Thus, they exploit the cooperative nature of the other users, without being willing to waste their resources for them. Another example of free-riding can be found in file-sharing systems. In such systems, the free-riders refuse to share files within the network (i.e., minimizing the cost of bandwidth and CPU cycle), but they participate as clients by performing download in files contributed by others.

The second type of adversaries refers to malicious entities [71, 73, 24, 69, 112]. Their objective is to take over the system for their own purposes (in this case they implement one of the techniques: Traitors, Front peers (or moles), Sybil attack, Liars) or to disable the system entirely (in this case they implement one of the techniques: Inauthentic, Collusion, Whitewashers, Denial of Service (DoS), Impersonation). Example of malicious users can be found in file-sharing systems, in which their purpose is to harm the systems by either mislabelling dangerous contents (e.g., virus, Trojan Horses, etc) or by corrupting the contents. Moreover, in MANETs malicious nodes misroute packets form other nodes. In the following list, we classify the malicious entities based on their behaviour.

Inauthentic: Malicious peers, who participate in transactions with different content from the agreed.

Traitors: Malicious peers, who behave properly for a while when entering into the network, in order to increase their reputation. After achieving that, they reveal their negative behaviour (they begin defecting). This type of behaviour causes more problems towards systems that give additional privileges to user with high reputation. This happens because the malicious peers, when defecting, are able to perform extra damage to the systems due to the privileges that they have got from the system. An example is an eBay merchant, who behaves well in transactions with low cost, in order to increase his reputation. When he has gained high positive reputation, the system will trust him for higher priced transaction and the merchant will defect.

Collusion: Malicious peers who participate in a collusion in order to cause more damage to a system, than each one acts independently. Each malicious peer of the collusion is fully aware of the target, and the system considers the collusion as a single unit.

Front peers (moles): Malicious peers who cooperate each other in order to increase their reputation. When achieving this, they provide misinformation in order to promote actively malicious peers to the network. This type of attack is difficult to be neutralized especially in networks, where there are no pre-existing trust relationships (e.g. from a direct transaction between the two peers) and the peers have to rely on the ratings of other peers.

Whitewashers: A peer reveals a malicious behaviour until his reputation gets very low. Then, he exits from the system and enters again using a new identity (pseudonym).

Denial of Service (DoS): In this type of attack a set of malicious peers request from a system a very large amount of resources or services, in order to occupy it fully, and, thus, the system's resources or services are not available to legal users. This attack is conducted at the application or network layer.

Impersonation: Malicious peers try to personate legal nodes in transactions in order to modify their reputation in the way they want. The malicious peers may also propagate dangerous files, without reducing their reputation values.

Sybil attack: In this type of attack the malicious peers appear with stolen or forged identities in the network causing undesirable results.

Liars: In this type of attack the attacking nodes notify misleading reports for their transactions. The effect is that the network infers wrong results about the level of the nodes trustworthiness.

Finally, we should mention that the number of selfish peers is likely to be far greater than the number of malicious participants in a system. This happens because it is natural for rational users to be selfish and try to maximize their own gains, while it is not natural for them to behave in a malicious fashion. As long as there are shared resources, some users want to use more than the fair share for their own benefit, even if this selfish behaviour is at the expense of the community. On the other hand, malicious peers are a more acute threat because malicious behaviour is more harmful than the selfish behaviour in an autonomic system.

4.3 Security requirements

Security requirements in the framework of autonomic communications focus on the protection of code, data and resources of a system. These requirements needs to be determined using global attributes (such as privacy, confidentiality, anonymity, integrity, accountability and availability) in order to specify the appropriate level of security. In the following, we present a common understanding of the security requirements in the framework of autonomic communications, and provide a brief overview of them.

An autonomic communication system may want to keep any private data stored on a node, carried by an agent, or exchanged between system components confidential. For this reason, autonomic systems must be able to ensure that their intra- and inter-system communications remain confidential. Another piece of information that a node and an agent may want to keep confidential from other components inside or outside the system is their location. In addition, they may not want to disclose which nodes they have had transactions in the past. On the other hand, however, the security policy of a node may not be willing to accept nodes or agents that have had transactions outside of an approved security domain. Finally, since audit logs maintain details of the nodes' activities, their contents must be carefully balanced with the nodes' privacy expectations. Access to them must be restricted to authorized administrators. In general, the collection and use of audit information must be well defined and should be understood by the agents visiting the node.

Except for confidentiality, an autonomic system should be able to provide nodes with anonymity. It should keep a node's identity secret from other nodes, but it maintains a form of reversible anonymity where it can determine the node's identity, if necessary and legal. There are many situations, however, in which the participants are unwilling to engage in transactions with anonymous counterparts. Purchasers of goods and services may want to protect their privacy by remaining anonymous, but credit agencies would not extend credit to anonymous consumers without being able to verify their credit history and credit worthiness.

A node should provide agents' integrity. It should protect agents from unauthorized modifications of their codes, states and data, and ensure that only authorized agents or processes have the right to carry out any modification on shared data. In addition, the secure operation of autonomic systems depends on the integrity of local and remote nodes. A malicious node can easily compromise the integrity of an agent, which visits it. It may make subtle changes in the execution flow of the agent's code and changes to the computed results that are difficult to detect. It may also interfere in transactions between agents that represent different nodes and tamper with the audit trails.

Not only agents are targeted by attacks that originate from a node, but also the opposite. For this reason, system access controls must be in place to protect the integrity of the node from unauthorized users and from network worms, trojan horses and computer viruses. The agents should not be allowed to violate the node's resources (e.g., files, network resources, etc) and they should have only restricted access to them. Finally, many autonomic systems require persistence e.g., in cases of a computer crashes. To satisfy this, it is necessary to have a secure execution layer that is implemented either on a secure or insecure operating system and hardware.

Each process or agent on a node must be held accountable for its actions. Accountability requires maintaining an audit log of security-relevant events that have occurred, and listing each event and the agent or process that is responsible for that. In order to do so, the agents or processes must be uniquely identified and authenticated. In addition to accountability, audit logs are necessary and valuable when a node must recover from a security breach, or a software or hardware failure. Since audit logs contain critical information about the system, they must be protected from unauthorized access and modification. Finally, measures need to be in place to prevent auditable events from being lost when a storage media reaches its capacity.

A node must be able to ensure the availability of both data and services to agents and other nodes. It must provide support for simultaneous access, deadlock management, and exclusive access as required. Shared data must be available in a usable form, capacity must be available to meet service needs, and provisions for fair allocation of resources and timeliness of service must be made. Moreover, a node should be able to detect and recover from system failures (software and hardware). On the other hand, agents may be programmed to find new paths, if a network goes down between some nodes, or they may duplicate themselves. These make feasible the survivability of agents that is a preferable feature for mission-critical applications.

5 Defense measures: State of the Art and Self-Preservation mechanisms

In this section we present traditional security measures that have applied to secure network operations. Ranging from basic defence schemes to more complex ones, we exploit the considerations introduced in the previous section where we introduced the concept of two different security models: hard or traditional security and soft security. Both compete to protect the network and are required to enable self-preservation of the CASCADAS system.

First we discuss the paradigm of Mobile Agent Technology, which has many similarities with the concept of the Autonomic Communication Element, components of the CASCADAS network. Then, we present solutions that have been applied to protect the network from unauthorized use of resources through access control. Following the discussion on protecting the network from external attacks, we introduce solutions for protecting the network from Denial of Service Attacks.

We conclude the section defining the concept of self-preservation in an autonomic communication system and we sketch some basic requirements for realizing a self-preservation based approach. Then, we present three case studies from the area of overlay networks to examplify our ideas to self preservation. In this context, trust and reputation management system have a key role to enable the creation of a *soft* Web of Trust inside a virtual community along with mechanisms derived from traditional Game theory and Mechanism Design theories to deal with rational nodes.

5.1 Mobile Agents Technology and execution of services

Mobile Agents Technology (MAT) is an emerging technology that is gaining field in the area of distributed computing, realizing the concept of autonomic communications. MAT offers a new computing paradigm in which a program, in the form of a software agent, can suspend its execution on a host computer, transfer itself to another agent-enabled host on the network, and resume execution on the new host. The use of mobile agents has many similarities with the use of ACEs, and brings some interesting advantages when compared with traditional client/server solutions. The most prominent are [58]: the reduction of network traffic, communication and execution state transparency, autonomous and intelligent execution, programming and communication flexibility, adaptability to specific conditions, life cycle management, robustness and fault-tolerance, and interoperability.

On the other hand, the main reason that restrains the deployment of MAT and possibly the proliferation of ACEs are security concerns. The agent approach increases complexity and, thus increases the variance of security threats and attacks. A host may have access to agents' data, state and context. In addition, a visiting agent may act maliciously on the host or other agents hosted by the latter. Thus, the agent and ACEs technologies are susceptible to complicated attacks originated from both the host and agents. To defeat these security threats, an agent system has to incorporate a security framework that protects both the host and the visited agents. This framework should be customized according to the specific requirements of the system and the applied applications, as well as the associated risks that the deployed environment presents.

5.1.1 Attacks on Mobile Agent Technology

The MAT and the ACEs are in inherent danger of different types of attacks. These attacks are often categorized as: denial of service, damage of a system, breach of privacy, harassment, and social engineering attacks [72, 38]. In the following, we give a brief description of each type of attack, which may take place in mobile agents systems.

Denial of service: Executing agents or processes, an adversary may overload resources or services e.g., by constantly consuming network connections or by overloading buffers to create deadlocks.

Damage of a system: A malicious entity may alter or destroy a host's files, configurations, security policy, hardware or a mobile agent's code.

Breach of privacy: A malicious agent or a platform may get access to private data and disclose them.

Harassment: A malicious agent or process may perform annoying repeated attacks e.g., display unwanted pictures.

Social engineering: A malicious entity may manipulate people, hosts, or mobile agents by using misinformation. For example, a mobile agent may request users' passwords under false authority of the system administrator.

In addition, there are two more complex forms of attacks, which include features of the previously described attacks.

Event-triggered attacks: The initiation of any of the previously described attacks is triggered by a specific event, such as time, location, the arrival of a specific person, etc. (i.e., Trojan Horse program).

Compound attacks: Using cooperating techniques, agents may collaborate with each other in order to commit a series of attacks. For instance, harassment may be used as a part of a social engineering attack with an ultimate aim the system damage or the breach of privacy.

5.1.2 Security framework: protection

To defeat the previously analyzed security attacks and satisfy the advanced security requirements of mobile agents systems, a set of different security measures has been deployed, recently. These measures are either based on conventional security techniques used in contemporary distributed applications (with the necessary enhancements to be applied in mobile agent systems), or specifically devised for controlling mobile code and executable contents (e.g., Java applets) [48]. In the following, the recently developed security measures that aim at protecting mobile agents systems are briefly presented. These security measures are classified in two categories: (i) the one that focuses on agent platforms and (ii) the other that focuses on mobile agents.

5.1.2.1 Agent Platforms

This section presents the security measures that protect agent platforms. They protect a host platform from malicious agents by employing two general approaches:

- Isolation of critical data.
- Assuring mobile code.

Isolation of critical data

The isolation of critical data approach isolates memory and restricts access to it, in order to maintain exclusive execution environments. The most prominent techniques that use this approach are the *Software-Based Fault Isolation* and the *Safe Code Interpretation*, which follow the so-called sandbox security model of Java.

Software-Based Fault Isolation, as its name implies, is a measure that isolates application modules into distinct fault domains. It uses a software technique called sandboxing, which ensures that untrusted code modules and unsafe instructions are executed in fault domains with virtual addresses. To achieve this, it modifies software at the level of instructions preventing access to crucial system resources (e.g., memory) outside of the sandbox. Access to these is controlled through a unique identifier associated with each domain. Regarding the classification of instructions (i.e., safe and unsafe), Wahbe et. al. [119] consider the write and jump instructions as unsafe. Small and Seltzer [109] also consider the read instruction as unsafe, since some devices change state when they read. Finally, instructions such as reset, which change the memory access privileges and disable interrupts, must be prohibited.

The basic idea of the *Safe Code Interpretation* technique has to do with the safe execution of commands. Specific security measures are applied in order to convert harmful commands into safe; otherwise the commands are not allowed to be used by agents. The applied security measures are either static or dynamic. Static type checking in the form of bytecode verification is used to check conditions like type correctness, no stack overflow or underflow, code containment, register initialization, object initialization, etc. The static analysis of the bytecode at the loading-time ensures the safety of the downloaded code. On the other hand, another way to guarantee these conditions is to check them dynamically (e.g., array-bound checks), while executing the bytecode. However, checking these conditions at run-time is expensive and slows down execution significantly.

Assuring mobile code

The second approach protects agent platforms by assuring the code of mobile agents, which visit them. It includes measure that ensure the authenticity of mobile code and agents, the existence of appropriate safety properties in mobile agents, the detection of any malicious alternation made by an untrusted platform to mobile code, and the record of agents' history.

The authenticity of mobile code and agents is achieved by employing *code signing*, which involves public key cryptography and provides digital signatures. A digital signature serves as a mean of confirming the authenticity of an agent, its origin and integrity [100]. It enables the platform to verify that the code has not been modified since the creator or a user signed the agent (the signature from a user indicates the authority under which the agent operates). It also enables the verification of the identity of the signing entity, but it does not guarantee that it is trustworthy. To achieve this, each platform that executes mobile code should maintain a list of *trusted entities*. If the signing entity is included in the list, it is assumed that it is trustworthy and that its code is safe. In this case, the mobile code is given full privileges; otherwise, the mobile code will not be executed (i.e., "black- and-white" security policy).

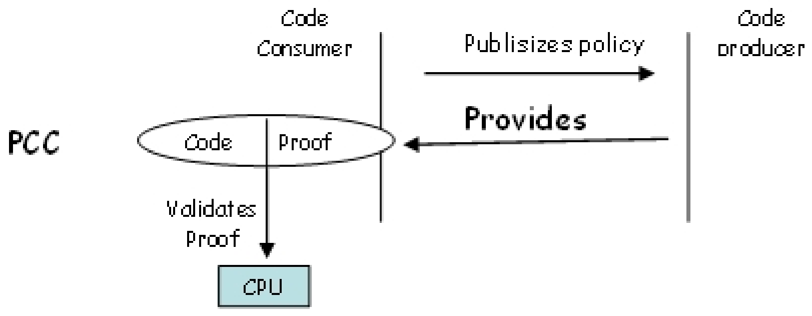

Figure 1: Proof Carrying Code (PCC)

The existence of appropriate safety properties in mobile agents is ensured by the Proof Carrying Code (PCC) [82]. This measure obligates the producer of a mobile code (i.e., the author of an agent) to formally prove that the program, written in a type-safe language (i.e., Java, C), possesses the required safety properties, previously, stipulated by an agent platform. The platform specifies a certain security policy that contains the conditions under which the execution of a foreign program is considered safe. Based on this policy, the producer constructs a safety proof (see Figure 1) from the smallest possible set of axioms and inference rules, in such a way that makes easy its verification without using cryptographic techniques or external assistance. The mobile code and the formal safety proof, which form the PCC binary, are is sent to the agent platform. The proof informs the platform about the existence of safety properties. The agent platform has also the ability to verify them.

Upon receiving the mobile code, a safety predicate, which represents the semantics of the program, is generated directly from the native code using a first-order predicate logic. This ensures that the companion proof does in fact correspond to the code (i.e., code certification). Then, the agent platform (i.e., the code consumer) uses a fast *proof validator-checker*, which conducts type checking and verifies the safety proof of the incoming code. Any attempt to tamper with either the code or the safety proof results in a verification error, which means code rejection. If the verification succeeds, the code is considered safe, and, consequently, executed without any further checking [48, 3, 61].

When mobile agents roam among platforms, they carry with them static data, collected data and execution state. The latter are dynamic data (i.e., program counters, registers, etc.) created and changed during the execution of an agent on a platform, and used as input to the computations performed on the next platform visited be the agent. A security measure that protects an agent platform and focuses on the execution state of the visiting agent is called *state appraisal*. This measure ensures that the agent has not become malicious or been modified due to alterations of its state at an untrustworthy platform [28]. The success of this measure relies on the extent to which harmful alterations to an agent's state can be predicted, and countermeasures, in the form of appraisal functions, can be prepared before using the agent.

The entity that creates a mobile agent produces a state appraisal function, which calculates the maximum set of privileges that the agent could request from a host platform. These privileges depend on the agent's current state, which is characterized by conditional factors and state invariants. Malicious states are defined via arbitrarily complex relations between dynamic state variables of the agent. The state appraisal function is actually a set of conditions that have to be fulfilled after one execution session. In addition, it may include some state invariants. Based on the predefined conditional factors and whether the identified state invariants hold, the appraisal function is used to determine what privileges to grant to the agent.

Similarly to the creator, the user, who sends the agent to act on his behalf, produces another state appraisal function depending on the current state of the agent and the tasks that are to be completed. The user packages the mobile code with the provided state appraisal functions and sends them to the target agent platform. Upon receiving them, the platform starts with the verification process. Depending on the verification results, the platform determines the privileges that will grant to the agent. In case that the agent violates an invariant, it may be granted with no privileges. On the other hand, if the agent fails to meet some conditional factors, it may be granted with a restricted set of privileges.

Finally, the path history [84] security feature maintains for each agent an authenticable record of the prior visited platforms. Using this record, a newly visited platform can determine whether to process an agent and what level of trust, services, resources and privileges should grant to it. These levels are estimated by simply reviewing the list of the identities provided by the path history or by individually authenticating the signatures of each entry in it. Recording path history requires that each agent platform should add a signed entry to the related record of the visiting agent, indicating its identity and the identity of the next platform that are to be visited. To prevent tampering, the signature of each new path entry includes a digest of the previous records [48]. The signing platform sends the mobile agent accompanied with the complete path history to the next platform. Depending on the information included in the path history, the latter may decide whether to execute the agent and what privileges should grant to it.

5.1.2.2 Mobile agents

As presented previously, the security measures directed towards the protection of agent platforms focus on active prevention, since mobile agents are completely susceptible to an agent platform. On the other hand, the security measures directed toward agents' protection emphasize more on detection, as an agent cannot prevent malicious behaviours from occurring, but it can detect them. In the sequel, we present the most prominent security measures designed for agents' protection, which also follow two general approaches:

- Isolation of critical data.
- Detection of attacks - intrusions.

Isolation of critical data

The isolation of critical data approach can also be used to device security measures that target to mobile agents protection. Data isolation is mainly applied to the results of an agent's actions by encapsulating them either in the host platform or in the agent itself. Data isolation/encapsulation is carried out for different purposes using different mechanisms: (a) confidentiality using encryption, (b) integrity and accountability using digital signatures, or (c) privacy using other cryptographic primitives like authentication codes and hash functions [129].

A mobile agent may encrypt the results of its actions using public key infrastructure. More specifically, it uses the public key of its originator (i.e., home platform) to produce small pieces of ciphertext, which are decrypted later at the home platform of the agent using the corresponding private key. For encryption the agent uses a special implementation called sliding encryption [48], which enciphers a small amount of data within a larger block, sliding away a small fraction. This fraction constitutes the ciphertext, which is derived from a state affected by all preceding ciphertext blocks. This security measure does not prevent malicious behaviour, but allows for certain types of illegal modifications and tampering to be detected. The main advantage of this measure is that it can be applied independently of the capabilities of the agent platform and the supporting infrastructure.

Another security measure that enables an agent to encapsulate the results of its actions is called Partial Result Authentication Codes (PRAC). PRAC provides cryptographic checksums (i.e., Message Authentication Codes - MACs) using symmetric cryptography. For each visited platform, the agent uses a secret key (associated to the host platform) to compute a MAC, on the results of its execution. The computed MAC and the provided results constitute the PRAC, which is sent to the agent's originator. The employed key in the MAC computation is included in a list of secret keys that the originating platform provides to the agent, before launching it. After the MAC computation, the employed key is deleted from the list of secret keys guaranteeing forward integrity to the provided results.

Symmetric cryptography is also used in the *environmental key generation* feature, which enables an agent to perform predefined actions when some environmental conditions are true [96]. More specifically, this feature is used in cases that a platform (i.e., host) wants to communicate with another platform (i.e., target) when an environmental condition is satisfied (e.g., activation of a specific procedure). Communication is achieved by a moving agent that carries encrypted (employing secret key cryptography) information (data and/or parts of executable code) and a method that generates the employed secret key using environmental observations. When the environmental condition is satisfied, the relative secret key is generated and the protected information can be deciphered. Otherwise, the carried information remain protected since the environmental condition is hidden using a one-way hash function. Thus, this feature ensures that the target platform or an individual observer cannot have access to the triggering condition and the protected information by directly reading the agent's code.

Isolation of critical data using cryptography is also applied in the *Computing with Encrypting Functions (CEF)* [102]. This measure allows an untrusted host to perform useful computations, without being able to identify the original function, and, thus, understanding what the computations are about. For materializing this, CEF discriminates functions and the programs that implement them. A program consists of cleartext instructions that a processor understands, but from these instructions the latter cannot understand the performed function. Employing CEF, a mobile agent can be executed in a potentially hostile environment, but the host cannot understand the encrypted functions of the agent. Thus, the agent can execute security sensitive functions (like signature functions), without the risk of being abused by the host, which may try to spy on the agent's code during execution.

Finally, *obfuscation* aims at protecting the code of mobile agents from being analyzed and understood by malicious hosts, using mess-up algorithms [43]. The basic idea behind this measure is simple: scramble the code in such a way that is difficult for anyone to get a full understanding of the program's function, or to modify the scrambled code without detection. Currently, there is a variety of obfuscating transformations: Layout Obfuscation, Data Obfuscation, Control Obfuscation, etc. Regarding performance, Layout and Data Obfuscation techniques reduce the size of the code, and, thus, speed up its execution, while Control Obfuscation results in the opposite [124].

Detection of attacks

In addition to the isolation of critical data approach, detection of attacks can also be used for protecting mobile agents. The security measures that follow this approach analyze the history of agents in order to detect the results of malicious actions, which have been performed against the agents. In the following, the most significant detection techniques that can be applied to mobile agents are presented and analyzed.

Mutual Itinerary Recording technique organizes the agents in groups of two (i.e., peers), and for each one maintains an itinerary record. The peer agents co-operate each other, exchange information about the host platforms and take appropriate action when inconsistencies are noted. The hosts are classified into red, grey and white based on whether they are malicious (red hosts), unknown (the agent does not have much information - grey hosts) or trustable (white hosts). All this information is included in the itinerary records of agents constituting their perception for each host. When moving between host platforms, an agent conveys information regarding the last platform, the current platform, and the next platform to the cooperating peer through an authenticated channel. The rationale behind this security measure is founded on the assumption that only a few agent platforms are malicious, and even if an agent encounters one, the platform is not likely to collaborate with another malicious platform being visited by the agent's peer. Therefore, by dividing up the operations of an application between two agents, certain malicious behaviour of an agent platform can be detected. Attention is paid so that an agent avoids platforms already visited by its peer.

A variation of the Mutual Itinerary Recording technique that supports fault tolerant capabilities is *Itinerary Recording with Replication and Voting* [47]. Applying this measure, multiple copies of the same agent are executed on different hosts to perform the same computation. The result computed by the majority of the hosts is accepted to be the correct. This security measure is employed under the assumption that the majority of the hosts are not malicious. Although a malicious platform may corrupt a few copies of the agent, enough replicates ensure the successful completion of the computation. Thus, the application of fault tolerant capabilities facilitates to counter the effects of malicious platforms. For each computation stage, the host platform checks whether the previously visited hosts are trusted, and ensures that the arriving agents are intact by checking the itinerary records. The host platform proceeds to the next stage, only, if a subset of the replica agents is considered valid. In addition, it propagates, only, the specific subset of replica agents acknowledged as valid.

Execution tracing is a security measure that detects unauthorized modifications to an agent using the faithful recording of the agent's behaviour during its execution (i.e., agent code, state and execution flow) on each agent platform. This measure requires that each host platform involved creates and retains a non-repudiable log or trace, which includes sequence of statement identifiers and platform signature information such as message's identifier, sender's identity, timestamp, etc. The recorded log for an agent is related to the operations performed by the agent while resident at a specific host. After finishing execution, the host produces a hash/fingerprint [117] of the log by applying a cryptographic hash function, and sends it to a trusted party or the agent's owner for verification. The hash of the trace and the related intermediate state are signed and forwarded to the next host platform in the itinerary. The signature of the host platform is necessary for instructions that depend on interactions with the computational environment maintained by the platform. On the other hand, the signature is omitted in cases that instructions rely only on the values of internal variables.

Upon return of the agent to the agent owner, a trace operation is triggered if the owner suspects that a certain platform cheated, while executing the agent This operation first asks from the suspicious platform to reproduce the related trace. Then, the agent owner starts a simulation of the trace beginning from the first host platform in the itinerary. This procedure will produce a trace identifying an intermediate state and the next host platform in the itinerary. The agent's owner validates the execution of the agent by comparing the reproduced-simulated trace with the hash of the trace and the intermediate state that he possesses. The process is continued detecting any derivation, and thus, identifying any malicious host. A variation of the above security measure is the *Execution Tracing with a verification server* [113]. This measure modifies the original by assigning the trace verification process to a trusted third party, the *verification server*, instead of depending on the agent's owner When a mobile agent travels to a new platform during its itinerary, a copy of the agent and a trace of its execution at every host platform is submitted to a corresponding verification server prior to the agent's migration to a new host platform. Each trace submission occurs within a specific time period checked by a time-out feature that this measure support. The submitted traces for all agents are maintained at the verification server. The latter simulates the execution of the agent by using the submitted trace and the copied agent, and, thus, detects any violations [16]. Therefore, in this security measure the verification process is not triggered only by suspicious results.

5.2 Access control

In autonomic systems nodes actively participate in the process of service creation and are no longer within the boundary of a secure and trusted domain. These concepts lead to a dynamic paradigm for service composition and mechanisms to control the access to resources. A first step in this direction consists of ensuring that only authorized nodes can access the resource or can contribute to its creation. Thus, security and negotiation of the access are tightly coupled: nodes must authenticate in a secure way and trust each other to access services.

Trust evaluation is performed by applying context-specific rules, metrics and policies on the trust evidence. The result of the process is the trust relation between the trustor and the trustee. Trust relations can be revoked on the basis of newly obtained evidence. Trust is transitive if it can be extended beyond the two parties, between whom it was established, allowing for the building-up of trust paths between entities that have not directly participated in a process of trust evaluation.

5.2.1 Trust Management

Traditional approaches rely security framework upon identity verification within a specific domain. Local administrative authorities sign/certify the association of a public key to an entity; the entity proves the ownership of the certificate by knowing the correspondent private key and the verifier checks the validity of the signature. The X.509 [39] framework provides means for building trust management using *certification authority trees* that issues certificates for the entity. If two entities wish to establish a trust relationship, they must have a common point (Certification Authority) in the certification authority tree.

The context of certificates in trust management is limited to the verification of the entity while access control is managed by using local application policies that checks the actions an entity is entitled to perform with respect to the local domain and to the rights of the issuer certification authority inside a specific domain. In trust management, common practice is to abstract the notion of certificate, the binding of a key to an identity, and to define digital credentials. These credentials also provide a description of the entity (e.g., rights, properties, attributes) along with identification.

Digital credentials and Policies

Digital credentials are digitally signed by using Public Key Infrastructure to ensure integrity and verification of the properties/attributes associated to the credentials. A credential works as access key to resources, which are protected by disclosure policies, and depending on how credentials are used, the access is referred to:

Identity-oriented access control This approach consists of two phases: authentication and authorization. Authentication requires the verification of the binding between the name specified in the subject field of a certificate and the public key. Authorization consists of the access decisions that can be taken on the access rights bound to the name. X.509 Public Key Infrastructure [39] can be used to implement the authorization stage, while the binding of the name with attributes can be implemented with attribute certificates and a X.509 Privilege Management Infrastructure (PMI) [45].

Key-oriented access control This approach integrates the authentication and authorization stages as the access control is performed on the attributes bound to the public key rather than on the name. In this case, the authentication part is accomplished by proving the possession of the private key associated to the public key. Two examples of a key-oriented access control are SPKI [26] and KeyNote [7, 8].

The set of credentials required accessing a resource are specified through policies. Once this set is determined and satisfied, additional credentials may only cause the disclosure of additional resources. This requirement, known as *monotonicity*, implies that the definition of the policies is crucial in trust management as the establishment of trust depends on their consistency.

Thus, a policy should be expressed in a well-defined and simple language which must be independent from the application language implementation. Weeks [120] proposes a semantic for trust management.

The verification of credential and the applied security policies restrict the access to the resources and services, and protect the owner from external attacks. Thus, in such systems it is implemented hard security allowing the access only to authorized users. On the other hand, a limitation of these systems is that they allow an entity to aggregate a perception of other entities, in order to choose the appropriate service/ resource. Thus, there is no need for the requesting peer to build trust in order to have access to the resources. Some examples of trust management systems are PolicyMaker [9], Keynote [7] and REFEREE [18].

PolicyMaker and KeyNote

In distributed systems, initial effort toward a definition of *trust management* has been accomplished by Blaze et al. [9, 8]. Their approach abstracts the trust management problem from the application context. Access control is delegated to a trust management engine, named PolicyMaker, that checks rights according to requests, local policies and keys specified in certificates. The authenticity of the request is delegated to the keys, as they are used instead of names or identities and access permissions are associated directly with keys.

A common language is defined to express authorization policies where conditions under which an individual or an authority is trusted and conditions under which trust may be deferred can be specified. In this context a credential, defined by using predicates, specify the authorized actions that the issuer delegates to the public key holder.

KeyNote [7, 8] is a trust management system derived from PolicyMaker. The design principles are the same but KeyNote implements a different approach for credential verification as it is done by the trust management engine rather than by the application. The application is in charge of constructing the so called *Action Environment* that contains relevant information for the request and necessary for the final trust decision. The action environment is defined as a set of attributes and values that reflect the security requirements of the application.

The trust management model defined by KeyNote and PolicyMaker does not handle Certification Revocation Lists (CRL). In the context of establishing trust, KeyNote and PolicyMaker do not address credential discovery as the language used for credential description ensures only the access rights delegated to the holder. These limitations do not guarantee that flexibility required to create trust from a negotiation between two parties involved.

REFEREE

Rule-controlled Environment For Evaluation of Rules, and Everything Else, shorted to REFEREE [18], is a trust management system based on PolicyMaker. REFEREE has been developed to regulate access to web application. The main corpus is the trust management engine that checks whether policies are satisfied and which access is granted to the requestor.

Like PolicyMaker, REFEREE defines a language for credentials and policies so that they are application independent, but it is also capable of signature verification like KeyNote and provides in addition credential discovery. However, the engine is more complex as it also supports applications to execute each other as subroutines. The result of a verification request can take three values: false, true, or unknown.

5.2.2 Trust Negotiation

In an open and distributed environment, trust management schemes fails short to accomplish incremental establishing of trust relationship between entities. Incremental trust relies on the assumption that entities want to protect not only resources but also their credentials or policies which might contain sensible information. Yu et al. [131] consider each policy that regulates access to a resource as a sensible resource. As a consequence, the access to the policy is managed by another policy in a recursive manner. Thus, an interactive process must be performed to enable entities at negotiating their trust relationship.

Trust negotiation consists of the bilateral disclosure of personal information between interacting parties in order to establish a trust relation for completing a transaction. Currently, trust negotiation is based on digital certificates which contain the attributes relative to a party involved in the communication. By using digital certificates, the two parties do not need to have a pre-established relationship in order to accomplish a transaction. Credentials are signed by the credential issuer who is responsible for the authenticity of the owners’ information contained in the credential.

In trust negotiation, policies regulate the exchange of information: access control or disclosure policies are the means to provide authorization to a party for accessing services. Credentials are exchanged upon disclosure of a policy in order to satisfy it. Credential flow is the basis of the trust negotiation and describes how credentials are disclosed along with an architecture for automated trust negotiation [121].

Credentials are disclosed according to the level of trust so far established and to the *trust negotiation strategy* [130]. A strategy controls the content of the messages such as the credentials submitted the disclosure of sensible policies and when terminate the negotiation. Each party may have a different local negotiation strategy and follows the *trust negotiation protocol* that regulates the exchange of messages. The choice of the appropriate strategy consists of a tradeoff between the caution of the party in disclosing credential and policies, and the cost (or speed) of the negotiation in terms of number and size of messages exchanged. Examples of trust negotiation systems are TrustBuilder [122], Trust-X [6] and iAccess [56]

5.3 Denial of Service

Of critical interest in the context of inter-ACEs communications are those threats that attempt to prevent legitimate users from using a given service. These threats are known as *denial of service* attacks. Denial of Service attacks have been known to the network research community since the early 1980s; indeed, in his 1985 paper on TCP /IP weaknesses [79], R.T. Morris writes *"The weakness in this scheme (the IP protocol) is that the source host itself fills in the IP source host id, and there is no provision to discover the true origin of the packet"*.

A typical generic DoS attack is practically always distributed (DDoS) [31]: the attacker uses security exploits in unpatched software, such as email applications or web browsers, to take control of a large number of lightly protected computers (for example those without a firewall or up-to-date antivirus software). The attacker can then order the compromised machines to simultaneously send a large number of packets to a specific target. As a result, traffic rates to routers and links in the vicinity of the target exceed their capabilities, and legitimate clients can no longer connect. Typical targets are the servers of e-commerce websites, which can suffer significant financial losses. Other targets may be news websites, corporate networks, or banks. The attackers exploit the weakness of IP by faking their source IP address (IP spoofing). This makes it difficult to defend against such attacks, or to trace the origin of the offending traffic.

Many different types of progressively more sophisticated DoS attacks have been used in the last several decades. Many exploit some specific vulnerability in a system [12, 115]. Most however simply involve sending the victim a larger number of packets or data that it can handle, resulting in either congestion, a crash, or in the worst case exposing some further exploit [13].

5.3.1 Defence

In most cases a complete protection architecture should include the following elements:

- ***Detection*** of the existence of an attack. The detection can be either anomaly-based, signature-based, or a hybrid of the two. In anomaly-based detection, the system recognises a deviation from the standard behaviour of its clients, while in the latter it tries to identify the characteristics of known attack types.
- ***Classification*** of the incoming packets into valid (normal packets) and invalid (DoS packets). As in detection, one can choose between anomaly-based and signature-based classification techniques.
- ***Response***. In the most general sense, the protection system either drops the attacking packets or redirects them into a trap for further evaluation and analysis.

Detection and classification usually overlap, since the methods used to detect the existence of an attack may well provide the necessary information to start responding towards probably normal and probably DoS packets.

5.3.1.1 The Packet Source

One of the simplest approaches to thwart IP address spoofing is to configure routers to drop arriving packets with IP addresses which are deemed to be outside a predetermined "acceptable" range [30]. It is currently undecided how the ACEs of CASCADAS will be identified. Determining the validity of the source, however, remains relevant and important in DDoS defence. This can be done passively, for example:

- Is the source of the packet a known "loyal client"?

- Did the source of the packet first appear before or after the detection of the attack?
- Is the client honouring his/her QoS agreements?

The above general passive tests have the advantage of being relatively light-weight, but are by themselves not sufficient to achieve accurate classification. More accuracy inevitably requires more specialisation.

In Mirkovic's et al leading work in [76], a set of anomaly-based classification criteria for flows and connections are proposed. For instance, a TCP flow may be classified as an attack flow if its packet ratio (TCP_{rto}) is above a set threshold. Similarly, for the ICMP protocol they propose to use $ICMP_{rto}$ as a detector. For the UDP protocol, a *normal flow model* is proposed to be a set of thresholds based on the upper bound of allowed connections per destination, the lower bound of allowed packets per connection, and the maximum allowed sending rate per connection. The classification of connections is also done based on limits of the connections' allowed packet ratios and sending rates.

One more option for validity tests is to use criteria based on the type of service. For example, a network which offers Voice-over-IP (VoIP) services should include specific classification criteria based additionally on the intricacies of VoIP traffic behaviour. Then, it becomes a matter of who has the most advanced knowledge in this case on the VoIP traffic behaviour, the attacker who tries to emulate it or the defence system of the victim which examines it.

The limits and thresholds used by all these types of tests can be set by using the network administrator's experience, or with an automatic learning process in all or some of the nodes, using data collected from on-going observation.

5.3.1.2 Active tests

The first question that one has to answer when under attack is whether it is a case of real attack or simply a case of unusually high legitimate traffic. That is because this ramp-up behaviour of traffic occurs also during *flash crowds*, when there is a sharp increase in the number of legitimate visitors to a website due to some significant event. Moreover, DoS attackers have recently started exploiting this fact by abandoning full strength bandwidth floods in favour of attacks which escape detection by imitating the signature characteristics of flash crowds. In response to this, detection mechanisms should try to distinguish between flash crowds and attacks after a ramp-up has been detected. Fortunately, there is a fundamental difference between the two. Flash crowd flows are generated by human users, while DDoS flows are generated by compromised computers. Thus, one can potentially use Reverse Turing tests to tell the difference. In recent years, Graphical Turing tests or visual CAPTCHAs, or simply CAPTCHAs (Completely Automated Public Turing Test to Tell Computers and Humans Apart), are commonly used to block automated requests to websites, such as the automated email account registration which has plagued Hotmail and Yahoo in the past. CAPTCHAs work by, in addition to entering their login name and password, requiring the user to copy a short piece of text from an image to authenticate themselves. A human can easily tell the sequence of letters that appear in a CAPTCHA 's image (even if it has been obscured by visual noise or written in an unusual font), while computers usually cannot. CAPTCHAs have been suggested to counter DDoS attacks against webservers [77].

A DoS detection mechanism cannot depend solely on CAPTCHAs. In all arms races it is only a matter of time for a countermeasure to arrive on the scene. Dedicated applications built by Computer Vision researchers achieve up to 92% success in solving commonly used types of CAPTCHAs [78]. Furthermore, in an autonomous system such as those proposed by the CASCADAS vision, many of the connections will be automated, meaning CAPTCHAs are of limited use.

Several methods have been proposed for actively challenging the clients' legitimacy, with Netbouncer being the most representative [114]. It keeps a list of authorised users (beyond suspicion), while the rest undergo a series of tests, divided into packet-based (for example a simple ICMP echo request), flow-based (originally with stateless TCP/SYN cookies) and application and session-oriented (such as CAPTCHAs). Various quality of service techniques are utilised to assure fair sharing of the resources by the traffic of the legitimate clients, while this legitimacy expires after a certain interval and needs to be challenged again. Although successful in most common cases, sheer volumes of traffic and the possible misuse of the identities of legitimate clients can overcome this type of defence.

Some of the above active test solutions may achieve impressive levels of detection accuracy, but all suffer from the common weakness of being exploitable as DoS vessels themselves, in the same way as the simplest form of active validation, the ACKs, are used as DoS vessels in the reflector DoS attack [87].

5.3.2 Detecting the existence of a DoS attack

Detection of DoS attacks would not be necessary in the ideal case of a defence architecture with proactive qualities that would render any DoS attack impossible. In fact, interesting proactive architectures have been proposed, but they do have some common disadvantages. Apart from the fact that to date no system is perfect, Denial of Service attacks against one's network do not happen very often and at least resource-wise a proactive protection system is usually too expensive to operate in the absence of an attack. This can be avoided if the protection system starts operating right after a potential attack is detected, which renders the detection step necessary in most realistic cases. In the following we present a short summary of the existing literature on detection methods.

5.3.2.1 Learning techniques

Jalili et al have proposed the use of an unsupervised neural network for the detection of DoS attacks [46]. They designed a Statistical Pre-Processor and Unsupervised Neural Net based Intrusion Detector (SPUNNID), in which a statistical pre-processor is used to extract features from packets. It is reported that the success of the scheme in recognising the attack is 94.5% and detection time is less than a second (0.7 seconds in the best case).

Gavrilis and Dermatas used radial basis function neural networks (RBF-NN) and statistical features to detect DoS attacks [34]. Their scheme comprises a data collector which collects the appropriate data fields from the incoming packets, a feature estimator that evaluates the frequencies for the encoded data and a RBF-NN detector for classification as either normal traffic or DoS attack traffic. For TCP traffic, the performance is evaluated with two different feature sets; the first set containing nine fields from the TCP header, and the second a subset composed of only three fields. The authors report that a correct classification rate of 98% was achieved at experiments with simulation data and detection success was as high as 100% for experiments with real network data. The results obtained by the three-feature set were similar to the rates obtained with the nine feature set. Experimental results reported for UDP were also successful, but with lower correct classification rate and smaller efficacy.

Gavrilis, Tsoulos and Dermatas studied the same optimum feature selection problem using genetic algorithms [35] and proposed a scheme for robust detection of DoS attacks. They present an effective method to determine which input features should be considered as being more important than others and which features have little or no relevance. They first select a set of 44 statistical features, and from this set, the optimum features to be used in detection are chosen using a genetic algorithm. The total scheme is composed of a data collector, a features estimator and a two-layer feed-forward neural network detector. The authors observe that selected features vary with the number of hidden neurons, and with the selection and mutation probabilities.

Noh et al [83] proposed a traffic rate analysis (TRA) mechanism to measure the TCP flag rate and the protocol rate and utilized three machine learning algorithms, namely C4.5 [92], CN2 [19] and a Bayesian classifier, for detecting DoS. They provided experimental results in a simulated TCP-based network. In TRA , packet collecting agents are used for classifying IP packets into TCP, UDP or ICMP packets and for further summing up the occurrences of individual flags for TCP packets. For a specific sampling period the rate of a certain type of flag is determined by dividing the total number of this flag's occurrences to the total number of TCP packets observed while the ratio of the number of TCP, UDP or ICMP packets to the total number of IP packets gives the protocol rate. The experiments showed that SYN and ACK flag rates for inbound traffic provided significant information for detection of SYN flooding attacks (as under normal conditions each SYN packet should result in an ACK packet) and the best performance was obtained by the Bayesian classifier. False alarms did occur but no missed alarms were observed.

Kim et al [54] proposed a data mining approach based on an automatic feature selection mechanism with a neural network classifier for DoS attack detection. They utilize a decision tree, together with entropy and chi-square (used to determine goodness of fit) concepts, to select the best features out of a set of candidate features. The selected features are used in the neural network classifier.

Siaterlis and Maglaris designed a data fusion system, with which they aggregated information about the internet traffic collected by different sensors [107], and they used the Dempster-Shafer Theory of Evidence [126] to combine the data. In the DS framework one can state hypotheses, define membership, belief, plausibility and doubt functions regarding the hypotheses and eventually combine all evidence using a rule, to obtain a single conclusion. The proposed system is tested on a university network where information was collected with a Snort [98] plugin and MIB entries.

He, Luo and Liu used an Adaptive Neuro-Fuzzy Inference System (ANFIS) together with the Fuzzy C-Means Clustering Algorithm (FCM) to detect DoS attacks and tested their method by performing experiments on a DARPA/KDD99 dataset [42]. Lee, Kim, Lee et al., proposed a scheme where first the content of the incoming packets are analyzed, then a probe detection system using fuzzy cognitive maps (PDSuF) is utilized to detect DoS attacks and a black list of IP addresses is constructed in accordance with the collected information [63]. In [81] Mukkamala and Sung applied three computational intelligent techniques; support vector machines (SVMs), multivariate adaptive regression splines (MARS) and linear genetic programs (LGP), in a multi-agent setting, to the problem of DoS detection and compared the experimental results obtained by all three methods. For these intelligent methods, Sung and Mukkamala also addressed the feature selection problem and gave feature ranking algorithms for each of them [111]. Cheong, Kim et al. performed a causality analysis by combining DoS attack types and network protocol measures in a cause-effect framework [17]. Giacinto and Roli et al., and Mukkamala and Sung et al. have used multiple classifier systems for intrusion detection and reported results for DoS detection [36, 81, 80]. In [15], Chan and Ng et al proposed an entropy based feature grouping method for multiple classifier systems and used RBFNNs as base classifiers. Shin, Kim and Ryu classified and minimised false alarms in DoS detection by using a decision tree approach [106].

5.3.2.2 Statistical signal analysis

Internet traffic has some statistical properties which can be used in the detection of DoS attacks, and several DoS detection schemes based on these statistical properties have been proposed [64, 65, 125, 40, 57, 57, 29, 44]. It is however unknown if these techniques will be applicable to an autonomous ad-hoc environment, without prior knowledge of the statistical properties of the traffic in such a system. Such schemes must be developed with hindsight.

5.3.2.3 Multiple agents

Peng and Leckie et al. [88] have proposed a multi-agent approach to detect DoS attacks. They devised the Source IP address monitoring (SIM) scheme, in which new legitimate source IP addresses appearing in the traffic are collected in the IP Address Database (IAD) in the off-line training phase. This database is utilized together with data from the incoming traffic to give decisions about possible DoS attacks in the detection and learning phase. A non-parametric change detection scheme, CUSUM (cumulative sum) is used to extract information about the abrupt changes in the number of new IP addresses. A threshold is then used to give decision about a DoS attack.

Each agent in the network applies this scheme and in case there is DoS suspicion, they will share their beliefs. Using gradient-based learning techniques, an optimum value of the threshold that minimizes both the communication overhead and the confirmation delay is obtained. Detection accuracy with this scheme is reported to be as high as 100%. This multi-agent approach will be more successful in detecting a highly distributed DoS attack, but it will be slower compared to a centralized system because of confirmation delays.

Cetnarowicz K., Rojek G. applied a multi-agent approach to the computer security problem with an emphasis on DoS attacks, by which they differentiated between good and bad behaviour in the network using concepts from biological immune systems [14]. Seo and Cho utilized an agent based scheme with a Black Board Architecture (BBA) and firewall to detect DoS attacks and to form a black list of IP addresses [104].

5.3.3 Responding to an attack

Proactive server roaming [53] is a novel idea that proposes that an active server changes its location within a pool of physical servers to defend itself against unpredictable or untraceable attacks. The important constraint in this approach is that only legitimate users are explicitly informed by the roamer of its IP address, and therefore are able to follow the active server as it roams. ACEs may move between physical locations in order avoid attacks in much the same way.

One of the leading works in the proactive defence against DoS is Secure Overlay Services (SOS) [52], which is geared toward supporting Emergency Services or similar types of communication. The architecture is constructed using a combination of secure overlay tunnelling, routing via consistent hashing, and filtering. It reduces the probability of successful attacks by (i) performing intensive filtering near protected network edges, pushing the attack point perimeter into the core of the network, where high-speed routers can handle the volume of attack traffic, and (ii) introducing randomness and anonymity into the architecture, making it difficult for an attacker to target nodes along the path to a specific SOS-protected destination. The goal of SOS is to route only the "confirmed" users' traffic to the server and drop everything else. The clients are authenticated at the overlay entrance and they use the overlay network to reach the server. Only a small set of source addresses are "approved" to reach the server, while all other traffic is heavily filtered out. SOS does not work for public service (the clients must be aware of the overlay network and use it to access the victim) and it still allows brute-force attacks on links entering the filtering router in front of the client.

The Mayday approach [4] generalises the earlier work on Secure Overlay Services [52]. Mayday combines overlay networks and lightweight packet filtering. The overlay nodes perform client authentication and protocol verification, and then relay the requests to a protected server. The server is protected from the outside by simple packet filtering rules.

Pricing techniques have also been suggested for protection against DoS attacks [70]. Dynamic Resource Pricing is a distributed gateway architecture and a payment protocol that imposes dynamically changing prices on both network, server, and information resources in order to push the cost of initiating service requests back to the requesting clients. This approach provides *service differentiation* and can therefore discriminate to some extent against adversarial behaviour.

A well-known latest effort in the field of DoS defence is DEFCOM [75], which suggests that the current paradigm of designing defence systems which operate in isolation should be abandoned. DEFCOM is a distributed framework that enables the exchange of information and services between existing defence nodes. Since, for example, attack detection is best done near the victim, while response is most effective and collateral damage is minimal at the source of the attack, each node should be specialised in a different aspect of the defence. Defence nodes must be able to communicate and must support at least the following messages: *Attack alerts* (generated from the Alert Generators to the rest of the network), *Rate-Limit requests* (the rate-limit requests should be sent upstream), *Resource requests* (each node should be able to issue a resource request to its downstream neighbours), and *Traffic Classification* (classifier nodes must communicate with their downstream neighbours to ensure that the bulk of legitimate traffic will not be dropped). DEFCOM is the scientific child of some of the most important active DoS researchers, but is still in a very early stage. For the time being, the only obvious limitation seems to be the fact that an overlay node could be compromised and damage the operation of a large part of the system.

5.3.4 Final remarks

The above systems and research are targeted at solving the problem of DoS attack vulnerability in the existing IP based network infrastructures. They are important in terms of the CASCADAS project as they highlight pitfalls to avoid when designing ACE communications architectures, and principles to apply to the ACE security architecture.

5.4 Self-Preservation

Self-preservation is an umbrella term used for describing techniques that aim at protecting the so called “utility” of an ACE from deterioration, owing, fundamentally, to the exposure of some of its resources to other ACEs, and entities in general. The term utility is loosely defined here as a measure of how successfully an ACE is performing the function that was intended to perform by the authority that has deployed it.

Self-preservation techniques can be thought to be position at a level below Denial of Service (DoS) defence techniques. The second aim at protecting from total loss of utility from massive attacks, whereas the first target less severe utility deteriorations owing to either coincidental (e.g., malfunctions, misconfigurations) or intentional (e.g., low intensity “parasitic” attacks [60]) circumstances.

Since self-preservation is concerned with utility issues in view of interactions between selfish agents with conflicting interests, it is natural to ask what is its relationship to Game Theory (GT) and Mechanism Design (MD). GT is used for predicting the outcomes of such interactions, whereas MD is used for creating the appropriate contexts that will lead selfish users to behaviours that are in alignment with the preferences of some “designer”, which typically assumes the role of leveraging and cultivating the so-call “social-interest” (or “good”, or “utility”, or “well being”).

This rich body of theory was born within the realms of economics and mathematics and through its development the last 50+ years has come to influence most scientific disciplines. The rise of the most complex computational artifact of our times, the Internet, has provided vast new fields for applying and pushing further the development of the theory [85]. Within such a complex system, both GT and MD have been used in two different styles: (1) a macroscopic one, aiming at predicting and designing the most essential high level properties of the Internet, and (2) a microscopic one, aiming at designing the detailed workings of the various elements that constitute the Internet. (1) is a natural adaptation of previous usages of the theory in domains like economics, political and management sciences, but (2) seem to be somewhat new. At a microscopic level, almost all of Internet’s mechanisms are rich in detail and complexity that relates to their specific functions (e.g., to route packets, to resolve names, to schedule and transmit, etc.).

Although notable exceptions exist and will un-doubtably continue to appear, it seems that in most cases both MD and GT are more at home with (1) rather than with (2). The reason is rather obvious: Both being theoretical tools, they have to employ assumptions and simplifications that are fundamentally at odds with the complexity and the over-specification that characterizes the network at the microscopic level. The threshold on the amount of precision and detail that can be “sacrificed” is sometime too low to allow for an effective use of the theory at this microscopic level. It seems that alternative approaches that do not have to be strictly bound to the requirements of the theory should be sought.

To address the aforementioned problem due to the inherently macroscopic approach of GT and MD, we will also introduce the so-called *Self-Preservation* mechanisms as an engineering approach, to bring some of the notions and the knowledge born in GT and MD into the microscopic level of complex systems such as networks of ACEs, in a way that reconciles the conflicting requirements for abstraction and precision.

In the next subsection we sketch some basic requirements for realizing a self-preservation based approach. This section is followed by a section that exemplifies our ideas to self preservation by presenting three case studies from the flourishing research area of overlay networks: the first example is on distributed replication and is based on the application of GT, while the other two are on caching and peer-to-peer streaming networks and exemplify more the engineering approach to self preservation.

5.4.1 A Sketch of the requirements for Self-Preservation

A mechanism or an entity that adheres to a Self-Preservation based approach is one that is *fundamentally open to cooperation, interaction, and sharing of resources with other such entities and mechanisms, provided that these do not harm its local utility* (or more accurately the utility of some rational owner(s) that control this entity). Seen from some distance, self-preservation is just an additional functional layer whose purpose is to remain quiet (inactive) as long as the interactions between other entities are beneficial and intervene by modulating the local behaviour only when these interactions lead to *mistreatment*, defined here as the utility level below which an ACE is considered to be under-performing. We target "open systems" in which group settings (*e.g.*, number of nodes, distances, demand patterns) change dynamically. In such systems it is not possible to address the mistreatment issue with predefined, fixed designs that result from "off-line" optimizations. Instead, we believe that *rational entities should adjust their scheme dynamically so as to avoid or respond to mistreatment if and when it emerges*. To achieve this goal we argue that the following three requirements are necessary.

Detection Mechanism: This requirement is obvious but not trivially achievable when operating in a dynamic environment. *How can a node realize that it is being mistreated?* In a previous work on replication [60], a node compared its access cost under a given replication scheme with the guaranteed maximal access cost obtained through greedy local replication. This gave the node a "reference point" for a mistreatment test. In that game-theoretic framework, we considered nodes that had *a priori* knowledge of their demand patterns, thus could easily compute their greedy local cost thresholds. In the sequel we will be looking at a similar application that employs object caching and assumes non stationary demand patterns. In such a setting the nodes have to estimate and update their thresholds in an on-line manner and construct a dynamic threshold for a mistreatment test. A generic promising approach for this is *emulation*. A node follows some specific behaviour and at the same time emulates other alternative behaviours (some more, some less cooperative) and switches between them when their relative performances with respect to the local utility changes.

Mitigation Mechanism: This requirement ensures that a node has a mechanism that allows it to react to mistreatment—a mechanism via which it is able to respond to the onset of mistreatment. In the context of the aforementioned caching application, this mechanism will be the object admission control mechanism used by the nodes to decide whether to cache or not an incoming object. As it will be discussed later on, nodes may avoid mistreatment by modulating the admission control according to the current operating conditions.

Control Scheme: In addition to the availability of a mistreatment mitigation mechanism, there needs to be a programmatic scheme for controlling the parameters(s) that affect this mechanism. In the case of admission control in caching, such a parameter is the probability with which an incoming object is allowed to leave a local copy in the cache as it goes through it. Since the optimal setting of these control parameters depends heavily on a multitude of other time-varying parameters of the environment (*e.g.*, group size, storage capacities, demand patterns, distances), it is clear that there cannot be a simple (static) rule-of-thumb for optimally setting the control parameters of the mitigation mechanism. To that end, dynamic feedback-based control becomes an attractive option (more in the sequel).

5.4.2 Self-Preservation by example

In this subsection we discuss briefly three examples to exemplify the CASCADAS approach to self preservation.

The first example shows how a traditional game-theoretic framework along with some low complexity algorithm design can be employed in solving the cooperation problem of selfish ACEs organized in a group to provide content replication services in an efficient way. That is, in a way that preserves the service efficiency of each individual ACE that would materialize had they not cooperated and acted in isolation, while at the same time it provides opportunities for additional efficiencies through cooperation. Since this CASCADAS work applies the traditional GT framework, which is well known, it is described very briefly, with the emphasis on the limitation of this work (and the GT approach in general), that call for the consideration of the engineering approach to self-preservation to supplement the game theoretic approach by applying it to more complex and dynamic environment. Details for this example may be found in the publication [60].

The other two examples illustrate the non-game-theoretic approach to self-preservation, referred to here as an engineering approach to self-preservation. The first example considers the management of content by cooperating ACEs in a dynamic environment that calls for caching rather than replication approaches to it. This example illustrates very clearly how a similar objective (i.e., providing efficiently content to users) can be pursued more realistically and effectively in a different environment. Details for this example can be found in the works [59] and [110].

The third example illustrates again the engineering approach to self-preservation, by considering a more demanding type of content provision (through p2p streaming) and discusses a number of issues that will need to be addressed in order for the ACEs to cooperate in an efficient maner and optimise the quality of the provided service. This work is in progress, with the preliminary ideas discussed in [116].

5.4.2.1 Distributed Selfish Replication

Consider a group of nodes that store information objects and make them available to their local users as well as to remote nodes. A user's request is first received by the local node. If the requested object is stored locally, it is returned to the requesting user immediately, thereby incurring a minimal access cost. Otherwise, the requested object is searched for, and fetched from other nodes of the group, at a potentially higher access cost. If the object cannot be located anywhere in the group, it is retrieved from an origin server, which is assumed to be outside the group, thus incurring a maximal access cost.

Previous Results under Replication

Under an *object replication* model, once selected for replication at a node, an object is stored permanently at that node (*i.e.*, the object cannot be replaced later). In [60] we established the vulnerability of *socially optimal* (SO) object replication schemes in the literature to *mistreatment* problems. We define a mistreated node to be a node whose access cost under SO replication is higher than the minimal access cost that the node can guarantee under greedy local (GL) replication. Unlike centrally designed/controlled groups where all constituent nodes have to abide by the ultimate goal of optimizing the social utility of the group, an autonomous, selfish node will not tolerate such a mistreatment. Indeed, the emergence of such mistreatments may cause selfish nodes to secede from the replication group, resulting in severe inefficiencies for both the individual users as well as the entire group.

In [60], we resolved this dilemma by proposing a family of *equilibrium* (EQ) object placement strategies that (a) avoid the mistreatment problems of SO, (b) outperform GL by claiming available "cooperation gain" that the GL algorithm fails to utilize, and (c) are implementable in a distributed manner, requiring the exchange of only a limited amount of information. The EQ strategies were obtained by formulating the *Distributed Selfish Replication* (DSR) game and devising a distributed algorithm that is always capable of finding pure Nash equilibrium strategies for this particular game.

The Limitations of Replication and GT and an Alternative Approach based on Self-preservation and Caching

Proactive replication strategies are not practical in a highly dynamic content networking setting, which is likely to be the case for most of the Internet overlays and p2p applications we envision. This is due to a variety of reasons: (1) Fluid group membership makes it impractical for nodes to decide what to replicate based on what (and where) objects are replicated in the group. (2) Access patterns as well as access costs may be highly dynamic (due to bursty network/server loads), necessitating that the selection of replicas and their placement be done continuously, which is not practical. (3) Both the identification of the appropriate re-invocation times [67] and the estimation of the non-stationary demands (or equivalently, the timescale for a stationarity assumption to hold) [49] are non-trivial problems. (4) Content objects may be dynamic and/or may expire, necessitating the use of "pull" (*i.e.*, on-demand caching) as opposed to "push" (*i.e.*, pro-active replication) approaches. Using on-demand caching is the most widely acceptable and natural solution to all of these issues because it requires no *a priori* knowledge of local/group demand patterns and, as a consequence, responds dynamically to changes in these patterns over time (*e.g.*, introduction of new objects, reduction in the popularity of older ones, *etc.*)

In [60] we were able to utilize GT effectively and prescribe Nash equilibrium replication strategies with all the aforementioned favourable characteristics (including ease of implementation). To do so, however, we had to consider a stationary situation. Persisting on the GT approach and trying to encompass requirements (1)–(4) seams infeasible. What is required in this case is an engineering approach that is power-full enough to meet the challenges of the dynamic environment implied by (1)–(4). At the same time, since our focal point remains on selfish entities, we still want to maintain the concepts and connotations of GT. This is exactly the setting in which we postulate a *self-preservation* based approach. In the sequel we demonstrate such a solution framework for *Distributed Selfish Caching* (DSC). We will be using caching as our enabling mechanism and modulate it under a self-preservation approach in order to achieve in a dynamic environment what replication and GT can provide only under a stationary environment.

5.4.2.2 Distributed Selfish Caching

Consider a similar environment as under the content replication environment above, with some notable differences indicated below. That is, consider a group of nodes that store information objects and make them available to their local users as well as to remote nodes. A user's request is first received by the local node. If the requested object is stored locally, it is returned to the requesting user immediately, thereby incurring a minimal access cost. Otherwise, the requested object is searched for, and fetched from other nodes of the group, at a potentially higher access cost. If the object cannot be located anywhere in the group, it is retrieved from an origin server, which is assumed to be outside the group, thus incurring a maximal access cost. The environment here is dynamic and objects are not stored permanently as in the previous example.

Causes of Mistreatments Under DSC

We begin our examination of DSC by first considering the operational characteristics of a group of nodes involved in a distributed caching solution. This examination will enable us to identify two key culprits for the emergence of mistreatment phenomena.

First, we identify the mutual *state interaction* between replacement algorithms running on different nodes as the prime culprit for the appearance of mistreatment phenomena. This interaction takes place through the so called "remote hits". Consider nodes v, u and object o. A request for object o issued by a user of v that cannot be served at v but could be served at u is said to have incurred a *local miss* at v, but a *remote hit* at u. Consider now the implications of the remote hit at u. If u does not discriminate between hits due to local requests and hits due to remote requests, then the remote hit for object o will affect the state of the replacement algorithm in effect at u. If u is employing Least Recently Used (LRU) replacement, then o will be brought to the top of the LRU list. If it employs Least Frequently Used (LFU) replacement, then its frequency will be increased, and so on with other replacement algorithms [89]. If the frequency of remote hits is sufficiently high, *e.g.*, because v has a much higher local request rate and thus sends an intense miss-stream to u, then there could be performance implications for the second: u's cache may get invaded by objects that follow v's demand, thereby depriving the user's of u from valuable storage space for caching their own objects. This can lead to the mistreatment of u, whose cache is effectively "hijacked" by v.

Moving on, we identify a second, less anticipated, culprit for the emergence of mistreatment in DSC. We call it the *common scheme* problem. To understand it, one has to observe that most of the work on cooperative caching has hinged on the fundamental assumption that all nodes in a cooperating group adopt a common scheme. We use the word "scheme" to refer to the combination of: (i) the employed *replacement algorithm*, (ii) the employed *request redirection algorithm*, and (iii) the employed *object admission algorithm*. Cases (i) and (ii) are more or less self-explanatory. Case (iii) refers to the decision of whether to cache locally an incoming object after a local miss. The problem here is that the adoption of a common scheme can be beneficial to some of the nodes of a group, but harmful to others, particularly to nodes that have special characteristics that make them "outliers". A simple case of an outlier, is a node that is situated further away from the center of the group, where most nodes lie. Here distance may have a topological/affine meaning (*e.g.*, number of hops, or propagation delay), or it may relate to dynamic performance characteristics (*e.g.*, variable throughput or latencies due to load conditions on network links or server nodes). Such an outlier node cannot rely on the other nodes for fetching objects at a small access cost, and thus prefers to keep local copies of all incoming objects. The rest of the nodes, however, as long as they are close enough to each other, prefer not to cache local copies of incoming objects that already exist elsewhere in the group. Since such objects can be fetched from remote nodes at a small access cost, it is better to preserve the local storage for keeping objects that do not exist in the group and, thus, must be fetched from the origin server at a high access cost. In this setting, a common scheme is bound to mistreat either the outlier node or the rest of the group.

In addition to the identification of the two causes of mistreatments in a DSC setting, this section summarizes a number of concrete results regarding each one of these two causes. A full exposition of these matters appears in Laoutaris et al. [59].

Mistreatment Due to Cache State Interaction

Regarding the state interaction problem, our investigations answer the following basic question: *"Could and under which schemes do mistreatments arise in a DSC group?"*. More, specifically:

- We have shown that state interactions occur when nodes do not discriminate between local and remote *hits* upon updating the state of their replacement algorithms.
- To materialize, state interactions require substantial request rate imbalance, *i.e.*, one or more "over-active" nodes must generate disproportionally more requests than the other nodes. Even in this case, mistreatment of less active nodes depends on the amount of storage that they posses: Mistreatment occurs when these nodes have abundant storage, otherwise they are generally immune to, or even benefit from, the existence of overactive nodes.
- Comparing caching and replication with regard to their relative sensitivities to request rate imbalance, we have shown that caching is much more robust than replication.
- Regarding the vulnerability of different replacement algorithms, we showed that "noisier" replacement algorithms are more prone to state interactions. In that regard, we showed that LRU is more vulnerable than LFU.
- Even the most vulnerable LRU replacement is quite robust to mistreatment as it requires a very intense miss-stream in order to force a mistreated node to maintain locally unpopular objects in its cache (thus depriving it of cache space for locally popular objects). In particular, the miss-stream has to be strong enough to counter the sharp decline in the popularity of objects in typically skewed workloads.
- Robustness to mistreatment due to state interaction evaporates when a node operates as a Level-2 cache [123] (L2) for other nodes. L2 caching allows all remote requests (whether they hit or miss) to affect the local state (as opposed to only hits under non-L2 caching), leading to a vulnerability level that approaches the one under replication.

Mistreatment Due to Use of Common Scheme

We can classify cooperative caching schemes into two groups: *Single Copy* (SC) schemes, *i.e.*, schemes where there can be at most one copy of each distinct object in the group – two examples of SC schemes are HASH based caching [99] and LRU-SC [27]; *Multiple Copy* (MC) schemes, *i.e.*, schemes where there can be multiple copies of the same object at different nodes.

- We have shown that the relative performance ranking of SC and MC schemes changes with the "tightness" of a cooperative group. SC schemes perform best when the inter-node distances are small compared to the distance to the origin server; in such cases the maintenance of multiple copies of the same object becomes unnecessary.[2] MC schemes improve progressively as the inter-node distances increase, and eventually outperform the SC schemes.
- We also demonstrated a case of mistreatment due to common scheme by considering a tight group of nodes that operate under SC and a unique outlier node that has a larger distance to the group. We showed that this node is mistreated if it is forced to follow the same SC scheme.

Towards Mistreatment-Resilient DSC Schemes

More constructively, in a follow up work (Smaragdakis et al. [110]) we presented a self-preservation based framework for the design of mistreatment-resilient DSC schemes. Our framework allows individual nodes to decide autonomously (*i.e.*, without having to trust any other node or service) whether they should stick to, or secede from a DSC caching group, based on whether or not their

[2]We do not consider load balancing concerns in this study.

participation is beneficial to their performance compared to a selfish, greedy scheme. Resilience to mistreatments is achieved by allowing a node to emulate the performance gain possible by switching from one scheme to another, or by adapting some control parameters of its currently deployed DSC scheme. We used a simple control-theoretic approach to dynamically parameterize the DSC scheme in use by a local node. We evaluated the performance of our solution by considering caching in wireless mobile nodes [127] where distances and download rates depend on mobility patterns. We showed that our self-preservation based schemes can yield substantial performance benefits, especially under skewed demand profiles.

5.4.2.3 Peer to Peer Streaming

Consider the case of large scale delivery of live video content. Due to the time sensitive and bandwidth consuming nature of live video, conventional client-server distribution systems provide limited capacity at high bandwidth costs while are unable to hold up against flash crowds. Native network layer multicast is a good solution but it is not widely deployed since it imposes great administrative costs requiring cross-domain cooperation. In this scene application layer multicast has emerged as a viable solution due to its ease of deployment, low cost of operation and scalability.

In application layer multicast an overlay network is constructed over the underlying existing infrastructure. The network’s nodes are unreliable and diverse end hosts (peers) who cooperate for an arbitrarily chosen time period, acting as clients and at the same time as servers for other peers. Peers are most commonly organized into a delivery tree optimized on the desired objective e.g. minimizing pair-wise delay or network cost. The key to providing service in this environment is cooperation. Each peer should pay its participation to the network by sharing resources with other peers. Limited shared resources may lead the system to saturation while on the opposite extensive sharing will permit the maintenance of a network with quality characteristics and high availability. Proper incentives should motivate participants to contribute resources to the network. Bandwidth is typically the critical resource. Peers exhibit bandwidth heterogeneity which can be both static, due to link speed differences, and dynamic, due to network congestion and errors. Furthermore each peer’s behaviour is unpredictable since it is free to join and leave the service abandoning the peers it serves. Disconnected peers will attempt to rejoin the application layer multicast tree but the effects of service interruption will be loss of video content and interruption of video playback unless robust techniques that recover from the effects caused by the dynamics of membership *a.k.a* churn, are deployed. A well-defined distribution system should provide mechanisms for: *(i)* Efficient overlay network formation and maintenance *(ii)* Cooperation assurance *(iii)* Heterogeneity management and *(iv)* Membership dynamics control.

Identifying the Causes of Churn

Previous work in this area [118] [74] aims to preserve video continuity of playback since lack of it is disturbing and frequently leads to non comprehensive video material. Video continuity is affected by peers’ decision to remain or depart from the system. The models typically adopted for a peer’s participation into the system are often quite simple and do not account for the realistic fact that a user has to be satisfied from the service it receives in order to continue its contribution to the system.

Results from recent peer to peer streaming trials indicate that there exists a strong correlation between the duration hosts stay into the system and the performance they observe [93]. Thus it is important to investigate how the experienced quality of service affects a peer’s lifetime in the system. Novel models capturing a peer’s participation into the service should be considered linking a measure of the experienced quality of service in terms of continuity of playback with the probability of the peer to leave the service. These models should also account for the fact that a peer may churn for arbitrary reasons.

In fact such models would reflect an attempt to take into account and predict behaviours developed in the social layer (*i.e.*, the users' domain) that have a direct impact on the dynamics developed in the physical layer (*i.e.*, the associated nodes' domain). It is expected that this view will greatly influence peer to peer streaming systems' design while it will provide a reasoning as to why presently deployed systems have not met the expected acceptance from the audience yet.

Self Preservation of Continuity

To preserve continuity in spite of the system's dynamics, a credit based scheme maybe employed aiming to always retain content ahead of playback time to be used during starving periods. Credit is the time period over which continuous playback at the end user can be supported without feed from the network. At the same time it serves as a useful metric of a peer's ability to cope with service deterioration in the future. Furthermore a new kind of peer "mobility" into the system is introduced other than the one caused by sudden disconnections due to the dynamics of membership. A peer whose credit value falls below a critical threshold value decides to move in quest of a serving peer potentially more able to support playback continuity.

The event of reaching credit's critical threshold may serve as a mistreatment detection mechanism. By reaching the credit's critical threshold, the peer infers that it is being mistreated in its current position in the overlay tree and most likely will not be able to support playback continuity in the near future, *a.k.a.* will not be able to preserve its local utility. This triggers a mistreatment mitigation mechanism which amounts to locating a new serving peer potentially more capable of preserving the peer's local utility. The mechanisms employed for selecting the most appropriate serving peer among the available ones, serves as the control scheme that modulates mistreatment mitigation mechanism.

Towards Mistreatment-Resilient Peer to Peer Distribution Systems

A peer's experienced quality in the past along with its potential to cope with service deterioration in the future are combined into a single metric capturing its expected behaviour. This knowledge is exploited along with other metrics of the quality of the communication in the construction of an overlay network with longer lived and stable overlay connections. Although most of the past work utilizes solely connection quality metrics, some papers relate a peer's age into the system with its expected behaviour [128]; exploiting this information in the overlay construction is expected to lead to longer lived connections while maximizing a peer's lifespan. A peer's age maybe a good metric be for more elastic services such as peer to peer file sharing but not so good for a service like live video streaming where participation serves no meaning if service at some time becomes discontinuous.

An ideally stable distribution network is the one that it is not affected by peer disconnections. This would require peers that are about to leave to be the leafs of the distribution tree so as their disconnection does not trigger the disconnection of the descendant peers as a side effect. Thus effective methods should straggle to make disconnections non perceptible to the end user. Since it is not possible to isolate disconnections due to the fact that this would require knowledge of each peer's lifespan, one could try to minimize the rate at which disconnections occur, *a.k.a.* the rate mistreatment occurs. This does not necessarily imply that the number of peers affected by a disconnection is minimal.

According to a selfish peer selection mechanism, each peer would select to join that serving peer who is on the path which is expected to live longer, in order to maximize the expected time until its unwanted disconnection from the tree occurs. Since a disconnection is likely to lead to a disruption in playback and increase its probability of a permanent departure from the service, minimizing the times the peer experiences a disconnection during its lifetime will minimize the probability for the peer to depart due to dissatisfaction during its lifetime.

The work on p2p streaming is on going. Preliminary ideas of this work may be found in [116].

5.5 Soft-management to assure Self-preservation

Any autonomic system such as the service-centred heterogeneous network proposed by CASCADAS must create and maintain trust in the system in order to function properly. Trust is an important component of all human (and machine) interactions whether they take place online or offline. The lack of any centralized administrative control makes the establishment of trust between participants difficult. There is no trusted central authority that records all transactions occurring in the system and creates a global view of the trustworthiness of the participants.

Trust management systems provide a sense of trust to nodes that are going to cooperate each other. Based on the approach of evaluation and establishment of trust, there are three categories of decentralized trust management system [37]: Policy based trust management systems, Reputation based trust management systems, Social based trust management systems. We refer to Section 5.2.1 for the first type of trust management system and we will present more in detail the reputation and the social based systems in the rest of this section.

5.5.1 Reputation-based trust management systems

Reputation-based trust management systems facilitate a user to choose the most trustworthy node. The goal is to establish trust relationships, which corresponds to specific values called trust values. The computation of a node's trust value is based on its reputation and the resources offered by him. Implementations of reputation-based trust management are XREP [22], NICE [62], P-Grid [1], EigenTrust [51], ROCQ [33, 32] (and others, such as SPORAS, HISSTOS, DCRC/CORC, Beta etc.)

Reputation management systems are tools for measuring trust in autonomic systems. They monitor participants' behaviour in all transactions in order to compute their trustworthiness. The peers' trustworthiness is determined either as an absolute scale or relatively to other peers. For the determination of trustworthiness, the system requires trust information gathered during transactions between nodes.This information is used by the nodes to build knowledge about the behaviour of other nodes, including nodes with whom they have never interacted before. Reputation Management Systems can be used for self-preservation of both participants and the services that they provide. The specific objectives of a reputation management system are [71, 50, 95]:

1. Facilitates nodes to decide whom to trust. A reputation system determines which of the nodes that offer a service or resources is the most trustworthy in order to provide to the requesting node the best quality of services or resources. Thus, the reputation system facilitates the node to decide which node to trust, in order to transact with. A necessary condition for correct decision is that the reputation system can determine which nodes provide the most reliable information (ratings).
2. Gives incentives, which motivates the nodes to positively contribute to the network. Positive contribution means that:
    - Nodes should cooperate when asked, showing trustworthy behaviour (that is they provide good quality of services or resources).
    - Nodes should provide reliable information (ratings).
3. Discourages the participation of adversaries. To achieve this, the reputation system identifies and excludes adversaries, who try to disrupt the system, and also those who provide unreliable information (ratings).

The above objectives should be served by a reputation management system with minimal overhead in terms of computation, storage and infrastructure. The reputation system should monitor user's behaviour in order to incorporate the necessary mechanisms that will help it to survive and not to collapse. In addition, it is essential for the reputation system to satisfy the requirements of users (e.g., must preserve anonymity associating a peer's reputation with an opaque identifier), in order not to disappoint and burden them, so that they would abandon the network.

To serve the first of the above mentioned objectives (i.e., facilitates nodes decide whom to trust), a reputation system implements three functions that provide gathering information (collection of trust information), aggregation of trust information (reputation score function or Reputation Computation Engine) and dissemination of trust information. In addition, in order to satisfy the second and third objectives, a reputation system implements two more functions, such as incentives and punishment. In the following we present and analyze the entire set of functions that a reputation system includes.

Gathering Information (collection of trust information)

Gathering information function collects information, which refers to the behaviour of a specific user in his previous transactions. The purpose of this function is to specify the reputation of the involved node (e.g. p1), based on the personal experience of the requesting node from previous transactions with the p1, and the recommendation of third parties. The gathered information, which represents the input of the reputation system, can be collected in a reactive, proactive, or hybrid way, based on the employed protocol. More specifically, if the gathering information protocol is implemented in a reactive way, then, the requesting node gets the information about the node that he is interested for when he needs it (on demand). On the other hand, if the gathering information protocol is implemented in a proactive way, then, the interested node gets information about any node before he needs it (i.e., the information is diffused either when a change is happened or periodically), and, thus, he is always updated. In these two approaches a trade off between delay for recovering information and overhead takes place. More specifically, the reactive schemes consume less bandwidth (lower overhead) than proactive, but they need more time (higher delay) for recovering information. The disadvantages that we meet in both approaches (i.e., proactive and reactive) make the growth of a hybrid approach necessary. In a hybrid scheme, the requesting node may receive in a proactive way information regarding the nodes that he is interested for (i.e., the nodes with which he has frequent transactions), and in a reactive way information regarding the nodes that are strangers to him.

The node's reputation is based on gathered information about it, from one or multiple resources. The selection of these resources depends on the protocol (technique) that the reputation system implements and comprises a trade off between the quality and the quantity of information about the node. An important issue in reputation systems is the assurance of trustworthiness (i.e. validity) of the ratings that are gathered. This happens because it is impossible to enforce all peers to be honest and to provide accurate report (rating) about the results of the transactions that they participated in. More specifically, the majority of the reputation systems do not try to confirm the honesty of the information they have gathered. On the other hand, some systems try to improve the accuracy of the transaction reports (ratings) by demanding a proof about the interaction that the report is referred to (e.g., such as TrustMe). In the following, we present the most prominent protocols (techniques) that are employed for resources selection in reputation systems [71], starting from this that emphasize more on information quality.

1st personal experience: The node trusts only his own experience, and, thus, he uses only local information. The reputation score is created only by the node's personal ratings and not by ratings that the node receives from other nodes. If all nodes in the network implement this technique, then, the function of trust dissemination would not exist. Personal experience is characterized as the best quality information, and for this reason it is weighted using the higher coefficient.

2nd external trusted sources: The node collects the opinion of other nodes with which he has external trust relationships. This means that there are priori trust relationships.

3rd one-hop trusted peers: The node collects the opinion of other nodes, which he has met in the network. These nodes may be all the nodes that he had a transaction in the past. An important issue is that the requesting node tries to increase the number of the reporting nodes increasing also the collected information. In case that the node wants to reduce the communication overhead, he asks for the opinion of the nodes that had provided to him good services (i.e. the respectable neighbour in the overlay topology).

4th multi-hop trusted peers: The node asks from his neighbours to collect the ratings of their own trustworthy neighbours. In this way it is formed a transitive trust chain (i.e. the trust transitivity principle is used), and thus, the sources of information are increased exponentially with the length of the chain (the base is the number of the node's neighbours in the overlay topology and the exponent is the number of hops between the node that asks for the information and the nodes that provides the information). The information provided in this way is probably more trustworthy than the information provided by a random node. The importance of the received information is weighted by the reputation rating using values between 0 and 1.

5th global history: In the global history reputation system, an agent collects information from all nodes about all nodes. This information is used for the computation of a reputation value by aggregating all ratings using weights. For example in the EigenTrust, the global reputation of each node is computed using transaction reports. These reports are weighted by the reputation of the nodes that give the report. However, this algorithm is vulnerable to collusion. To overcome this limitation, after the computation of the global reputation of a node, each node that is going to use this value weights it with his personal opinion about the node that it is going to have a transaction with.

Aggregation of trust information

When the collection of trust information has been completed and the rating weight has been estimated, then, the reputation score can be computed. More specifically, the history that has been gathered (i.e., the rating with the accompanying information, the cooperation degree, the defection degree, the time that the transaction has been done, the importance of the transaction, etc.) is the input in the reputation score function (also called as Reputation Computation Engine) that is implemented in the reputation system. The output of this function can be a binary value (e.g. trusted or untrusted), a scaled integer (e.g. 1 - 10), or on a continuous scale (e.g. [0, 1]).

Computing the reputation score, the first objective of a reputation system has been accomplished. As we mentioned above, this objective helps the nodes decide whom to trust. To achieve this, the requesting node must compute a reputation score for each node that is available to provide the desired service or resource. Then, it uses the reputation score of each respondent (potential service provider), in order to classify them according to how possible is for each of them to provide the appropriate service. After that, the requesting node selects the first from the list of candidates. If the transaction fails, the node tries again with the next from the list. There is also a possibility that the requesting node rejects all the offers of the respondent, if their reputation scores are under a predefined trust threshold. It should be mentioned that the existence of a threshold is necessary, in order to protect the requesting node from malicious respondents. The determination of a threshold creates a trade off between the quality and quantity of the nodes that are responding (and offering to provide their resources) and depends on how essential is the transaction for the requesting node. This means that if the transaction is considered essential and expensive, then, the node will put a high threshold in order not to be cheated. Otherwise, if the transaction is not considered to be essential, then the node puts the threshold lower in order to accept a greater number of offers.

Dissemination of trust information

The reputation system of each participant disseminates trust information about the behaviour of the nodes that the participant had transactions in the past. This information contains attributes of the nodes' behaviour during the transactions as well as some components of the transactions e.g., timestamp, etc. These attributes provide input to the reputation score function of the requesting node in order to calculate the reputation scores of the nodes. The disseminated information should follow a predefined format that is used in the reputation score function of the requesting node. If, for example, the reputation score function of a node weights the input information according to its antiquity, then the nodes that disseminate trust information must announce the timestamp of the transaction. Otherwise, in case that the reputation score function does not take into account the antiquity, then, the timestamp of the transaction isn't useful and it is considered as overhead to the network.

Incentives

The incentive schemes encourage the cooperation among nodes by trying to eliminate selfish behaviours. Selfish peers consume resources from the system without offering any resources of their own resulting in resource shortage. Giving incentives encourage nodes to cooperate because the cooperation cost is counterbalanced with some privileges that are offered by the incentive. Thus, selfish peers, in order to gain the privileges of the incentives, ignore the cooperation cost and, in this way, become useful by providing their services/ resources to the network. The most prominent types of incentives are either the provision of improved service or money. The provision of improved service concerns three general categories: a) speed or response latency, b) quality, and c) quantity.

The incentive schemes are effective against selfish nodes, but not against malicious. This happens because selfish nodes have the goal to maximize their own utility (without wanting wittingly to harm the network), while malicious nodes have the purpose to disable part (i.e. a specific participant) or the system entirely. However, there may be a form of incentive that mitigates malicious behaviour. For example, the system may force the users (malicious as well) to provide first good resources in order to have access to the system's resources. Such a reciprocal procedure increases the cost of bad behaviour (since the malicious users have to offer resources to the system, something that they would otherwise never do), while it doesn't affect the well behaved users at all (since they would offer sooner or later resources to the system). Finally, another effective solution to deal with malicious modes is imposing punishment.

Punishment

Reputation Management Systems recognize malicious nodes by their low reputation values, and, thus, they can deal with them. A primary function of the system to confront with the malicious nodes is to inform the users about them. This notification is for the benefit of the well behaved nodes because in that way they can avoid malicious nodes. Another way of dealing with malicious nodes is imposing punishment [71] (retaliation). The first form of punishment is ejecting malicious from the system. More specifically, the neighbours of a malicious node in an overlay topology can be disconnected from him, and, thus, ejecting the malicious node from the network. A second form of punishment is kicking of malicious nodes from the network for a period of time or permanently. If a malicious node is kicked for a period, to re-enter the system he should acquire a valid identifier. However, the identity system should provide a valid identity only if the malicious node has previously deposit a large amount of money, thus, the Whitewashing actions can be avoided. In this way, the frequent exit and re-entering the system is expensive or even impossible. Finally, a third form of punishment is the imposing of a fine to malicious nodes for each affirmed bad behaviour. However, such a solution should be used carefully, because a malicious node may implicate spuriously a well behaved node (e.g., through a collusion in which all conspirators will blame a specific well behaved node as malicious, in order to harm him). Therefore, this form of punishment can be used against the well behaved nodes and against the whole system.

5.5.1.1 Architectures of reputation management systems

The reputation management systems are implemented in two basic architectures: the centralized and the distributed. These two architectures differ on the way they compute the reputation scores and distribute the ratings. On the other hand, the architectural and implementation details of the aggregation mechanism depend on the underlying network on which the system functions.

In case that the community is built on top of a traditional client-server network, a trusted third party collects and aggregates opinions to form a global view. Examples of such systems, which store feedback from users in a centralized database, are: e-Bay, Amazon and Slashdot are examples where feedback from users is stored in a centralized trust database. In such systems the aggregation is performed in the centralized database and all the users have access to the global reputations computed. More specifically, in a centralized reputation system, the central authority (also called as reputation centre) collects the ratings regarding the performance of the nodes that participate in transactions. The central authority computes a reputation score for each participant that derives from the collected ratings and updates it when receives new inputs. Thus, it contains always updated reputation scores, which are available to all participants that use them in order to decide if they will cooperate with a specific node or not.

By implementing the reputation functions of a system in a unique trusted entity, the mechanism design is much simplified, and, consequently, a more efficient system is provided. On the other hand, there are many disadvantages, since it is rather difficult to find an entity that all nodes trust, and that this central entity might become a bottleneck. In addition, this unique serving point may become a target of attack by adversaries (it is especially vulnerable to DoS attack). In case that this central authority fails due to an attack, there is a risk that the entire network breaks down (the network looses its robustness).

In case that the community is built on top of a peer-to-peer architecture, as in autonomic communication systems, the challenges of managing feedback become much harder. There is no centralized, trusted, reliable, always-on database and the collection, storage, aggregation and dispersal of trust information must be done in a distributed way. For this reason two alternatives approaches are analyzed bellow. The first approach refers to the distributed storing using caches. After completion of a transaction, the participated nodes provide ratings about each other performance during the transaction, which are stored in a cache. A node, which wants to decide if it will transact with a specific participant or not, must implement a distributed searching protocol in order to find the distributed storing caches and acquire the ratings that refer to this participant. Then, the node must implement a reputation computation engine, which computes the reputation score of the target party, based on the ratings that it has obtained.

The second approach involves the nodes that participate in a transaction to store locally the ratings about each other performance during this transaction. Each node provides this locally stored information when requested from another node, who asks information in order to decide if it will transact with a target node or not. More specifically, each node implements locally a distributed communication protocol, so that it can receive as many ratings as possible by the nodes that had a direct experience with the target party previously. Due to the distributed environment, it is usually impossible or too expensive to obtain the ratings form all the transactions with the specific target node. It is worth noting that the global history is implemented only in centralized architectures. On the contrary, the distributed architectures use a subset of ratings that refer to target node. A technique that is easily implemented in distributed architectures and combines well the quality and quantity of ratings is the one-hop trusted peers. In this technique, the requesting node is based on the ratings that provide its neighbours in the overlay topology. After obtaining the rating values, the requesting node implements locally a reputation computation engine, which computes the target nodes' reputation score. However, relying on third parties for storage and dissemination makes the system vulnerable to tampering and falsification of trust information in storage and transit. In addition, the system must provide redundancy because users may drop out of the network at any time.

5.5.2 Social based trust management systems

The last category of trust management systems that is studied in this report refers to the social based trust management. These systems utilize the social relationships between nodes in a network to determine trust and reputation, so that trust relationships can be built. Therefore, there are many similarities between the social based trust management and the reputation based trust management systems. The main difference between them is that in the first a peer-to-peer network is mapped as a social network graph, which relies on various parameters such the weights on edges, the number of edges entering or leaving out of a node, etc. The view of trust that a node has in the network depends on its relative location towards the node who wants to compute the trust. Examples of such systems are Trustnet [103], NodeRanking [91] and Regret [101].

6 Roadmap

The CASCADAS project aims at strengthening and advancing scientific and technological excellence in the area of autonomic communication systems. This objective is achieved through the identification, development, and evaluation of architectures and solutions based on a general-purpose component model for autonomic communication services based on the ACE concept.

The new service-centric paradigm being proposed by CASCADAS introduces new challenges and security problems as outlined in Section 3.3. Security and self-preservation will make the network robust against attacks from malicious or malfunctioning entities. Entities that exhibit rational and irrational behaviors constitute the CASCADAS system and the system itself has to face new form of attacks and threats (see Section 4). This introduces new security requirements to the system such as monitoring activities, self-healing, distributed security policy enforcement and dstributed trust management. In the context of autonomic communication system (see Section 4.3), current solutions, presented in Section 5, solve only partially all the open issues brought by these new security requirements.

Thus, in the remaining of this Section we define clearly security in the context of CASCADAS and then we outline the approach we take to handle security principles founded on autonomic principles.

The resulting security framework is based on off-the-shelf technologies available at this time such as public-key cryptography, certificate authorities and digital signatures. We then extend this architecture by taking into consideration the challenges provided by the distributed nature of CASCADAS to provide self-preservation mechanisms. The outcome is the Security Architecture that we will present in Section 7.

6.1 Security in the framework of CASCADAS

The network envisioned for the CASCADAS project is based on the autonomic composition and aggregation of Autonomic Communication Elements (ACEs) to provide networking services in an adaptive and opportunistic manner. Autonomic communication services are characterised by the involvement of heterogeneous entities without centralised organisation or control, ranging from single users to entire institutions, to coalitions of institutions, and so on. These entities may actively contribute to the service definition/evaluation and self-management as well.

Autonomic Communication Elements (ACEs) offer up their resources (content, access bandwidth, storage and CPU) and services for the benefit of other ACEs that are part of and at the same time use the system, in order to reach a common goal, i.e., to provide an optimized value-added service. Thus, security requirements in the framework of autonomic communications focus on the protection of these resources. These requirements needs to be determined using global attributes (such as privacy, confidentiality, anonymity, integrity, accountability and availability) in order to specify the appropriate level of security. To deal with the complexity of the real-world, autonomic services, which involve large dynamic communities of users, should guarantee basic security functionality in order to enable secure interactions and at the same time provide advanced security services enabling soft-management and self-preservation of the entire system.

The decentralized approach and the freedom, that is required to form dynamic coalitions for service provision, increases the need for security as for the evolution of existing services the integration of existing components is dependant on the behavior of each entity. The lack of a central organization to manage the system and this dynamic creation of services require secure and flexible interaction among components which may not have established implicit and a-priori trust relations. As natural consequence, known security vulnerabilities in distributed systems must be tackled in addition to securing the new service-centered network proposed by CASCADAS.

There are inherent security services, required to create a world of trust to enable autonomic aggregation of ACEs and service provision, such as authentication, message integrity and confidentiality, non-repudiation and identity management and trust relationship through digital signatures and credential management. These security principles and algorithms constitute the basic building blocks to provide communication and information security.

However, the higher degree of "freedom" that is required for achieving self-organization has to be coupled with a higher degree of "responsibility" in order to assure the service survivability and self-preservation in the presence of possible malicious and/or malfunctioning entities. Due the uncertainty in the behaviour of the nodes we cannot relax security on the assumption that the protection of the communication or the availability of the resources are sufficient to guarantee a secure system. Thus, it is required to have in place other scheme to self-preserve the system and to guarantee its survivability.

6.2 Autonomic approaches for self-preservation in CASCADAS

The autonomic paradigm addressed by the CASCADAS project targets networks that can be viewed as distributed systems composed of autonomous entities, which need cooperation in order to work properly. The impact of cooperation or lack of it influence the performance of the system. In a heterogeneous environment, ACEs sometimes will deviate from a suggested protocol in order to improve their own outcome. As individual node interests might be in conflict with the network goals, nodes being selfish look first at their own interests when interests are in contrast. To model this situation, the approach that will be followed is based on Game Theory for predicting the outcome of interactions between selfish nodes and on Mechanisms Design to create the appropriate contexts that will lead selfish nodes to behaviours that will allow the system goals to be reached.

To give a concrete example of how Game Theoretic (GT) modeling is applied in the CASCADAS project, as well as to provide an overview on how the interaction between WP4 and other work packages of the project, we will focus on a fundamental component of the CASCADAS system architecture, that is the self-aggregation mechanisms developed in WP3 [21]. In CASCADAS, nodes should be able to self-organize into a stable overlay network defining logical connectivity among participants to the service oriented architecture. To do so, WP3 proposes a series of distributed algorithms able to foster aggregation between nodes sharing the same type, where a node type indicates the particular instance of a service that node is able to execute [2]. Jointly with WP3, we developed a mathematical framework that models the *selfish* interactions among overlay nodes when they are asked to re-wire an existing (and possibly random) overlay so as to make it possible for the overlay to absorb an un-even request load of users connected to the overlay. In particular, we present a novel formulation of what is traditionally called a *network creation game* wherein nodes (that is the players of the game) selfishly optimize a given objective function that describes the costs for a node of being connected to the service overlay. The cost function is designed so as to obtain a selfishly constructed overlay network that is connected, that is prone to support a non-balanced request load experienced by each node, and that at the same time minimizes communication costs, in terms of a physical metric such as number of hops traversed by a message or the latency experienced by messages to reach a destination.

While we address several issues related to the complex nature of the problem we also investigate on the impact of illegitimate behavior of nodes participating to the aggregation phase. Hindered by the computational complexity of the GT formulation we were constrained to come up with a simple and distributed heuristic that achieves the same aggregation objective as the one defined with the GT model. We then defined several classes of node misbehavior and analyzed their impact on the performances, in terms of aggregation efficiency, of the proposed distributed heuristics.

This work has opened several interesting new research issues such as designing incentives for node to correctly execute the aggregation algorithms, or more refined GT models that could take into account also the load partitioning among nodes executing the same instance of a particular service. These topics represent our future objective for the CASCADAS project, that we will tackle with a similar spirit of interaction and integration among WP3 and WP4.

Another approach for the continual self-preservation of both the ACEs and the services they provide in the field of soft security mechanisms, which is also called social control mechanism, is based on trust and reputation systems. Trust and reputation systems are useful tools for deriving provision trust, protecting in that way the relying parties from malicious or unreliable service providers. Provision trust is users' knowledge about the reliability of authenticated parties, or the quality of goods and services that they provide. This reputation system will rely on the dissemination, throughout the network, of trust information gathered through transactions between ACEs. In this way ACEs can build knowledge about the behavior of other ACEs (with whom they may never have interacted before) and the available services. This information can then be used to decide whether to interact with another ACE and whether to continue supporting an existing service or to start supporting a new service.

In this context, we exploit the capabilities of the nodes to self-preserve and to sustain the network through self-management systems. The system needs to be protected from external and internal attacks that mine the availability of the resources. For instance, such attacks can fall within the categories of Denial of Service attacks. Security countermeasures will be provide to protect each node, which should be able to detect the attack and respond autonomically to the threat. We rely entirely on the nodes to evolve the network and to generate and provide autonomically security services for self-preservation of the system. As a consequence, the security architecture envisioned for CASCADAS exploits the different functionalities of the nodes to provide security protection.

Some of these approaches have been already investigated in literature as presented in Section 5. However, the new general-purpose component model for autonomic communication services based on the ACE concept envisioned in CASCADAS require further investigation to address security issues and enable self-preservation of the system. Thus, as described in Section 5.4 we have taken an engineering approach for *Self-Preservation* mechanisms to bring some of the notions and the knowledge born in Game Theory and Mechanism Design into the microscopic level of complex systems such as networks of ACEs.

However, the activation of self-preservation mechanisms still requires a traditional secure communication framework. In the next Section we define the Security Architecture which is the starting point to enable the applicability and the deployment of self-preservation mechanisms on top of basic cryptographic functions. Thus, the anticipated activity consists in the integration of the security infrastructure with new features in order to fit the CASCADAS dynamic and autonomic network, which lacks of any central control point.

7 CASCADAS Security Architecture

This section outlines in detail the security architecture designed for the CASCADAS envisioned system. The architecture can be considered in a preliminary stage as its feasibility and applicability must be tested on the CASCADAS framework which has to be defined completely. But it considers the security requirements for applications and for autonomic communication systems as specified in Section 3 and Section 4.

The main purpose of the security architecture is to construct a trust framework so that components can authenticate and interact in a secure way. In an autonomic and distributed environment we cannot rely on a central authority or on a pre-defined secure infrastructure. Thus, trust relationships must be determined based on ad hoc security relationship which can take advantage from interactions established in the past or by means of a third temporal trusted entity or on opportunist and ephemeral way. By opportunistic we refer to the capability of the ACE to compute the risk of the new interaction without complete security assurances. The reasoning on whether participate in a transaction/aggregation (i.e. build a trust relationship) might consider the value of the service the ACE is going to access/provide or the context of the communication. The system is higly dynamic and ACEs' interactions might last a very short time, hence, the ephemeral context of the communication might not enable the definition of a durable trust framework that can be exploited to secure the system.

We foresee that ACEs form dynamically a virtual community and within this community ACEs might evolve and interact in the system or contribute in service provisioning as an aggregate component. Before the aggregation takes effect, components must interact and authenticate to create such virtual community that represents the ad hoc trust framework, or communication domain, which functions as support for ACEs during service provision.

However, in a high dynamic system composed by heterogeneous entities it could not be guaranteed that components' interactions and their subsequent aggregation are accomplished in a secure way. Cryptographic operations might require extensive computational capabilities or continuous message exchanges that are not proper of every ACE. The CASCADAS network also consists of tiny devices which have limited computational capabilities and limited connectivity or battery constrains. Furthermore, as the envisioned CASCADAS network is service-centered, specific applications or the context of the communication might not demand security features. Thus, secure aggregation might not be required in some cases or might fail if there are no means for its activation.

Then, we can conclude that, as it is not mandatory for ACE to interact in a secure way, we define security as a service that some ACEs can provide to the network. By considering the structure and the model foreseen in WP1 for an Autonomic Communication Element (ACE) [25], we assume that security services are placed in specific ACEs which implement basic functionalities. Security services can be defined as cryptographic operations or algorithms as well as advanced services for self-preservation and self-management of the system. We will clarify this distinction in the next Sections as it is the core of the CASCADAS security architecture.

The fact that security will not be considered as a particular service in the network justifies the assumption we made that security will not always be present in an aggregated component and that it will be added on demand if a suitable ACE component is found in the network.

In the next sections, we define the different types of nodes which constitute the CASCADAS network and their security capabilities. This part is fully integrated into the model designed for the ACE component while the definition of the secure aggregation and its success is demanded to the one for services which is out of the scope of the security definition and architecture at this point.

7.1 Security capabilities of ACEs

The heterogeneity of the CASCADAS system implies the analysis of the nodes that constitute the distributed infrastructure. According to the definition given in WP1 [25], an ACE is characterized by the job and functionalities it can offer to the network. These are defined by means of the specific part, see Figure 2, which contains specific features that specify the functions of an ACE.

Since security has been included in the CASCADAS framework as a service that will be used in aggregated form upon request, we specify the security service in the specific feature using the same

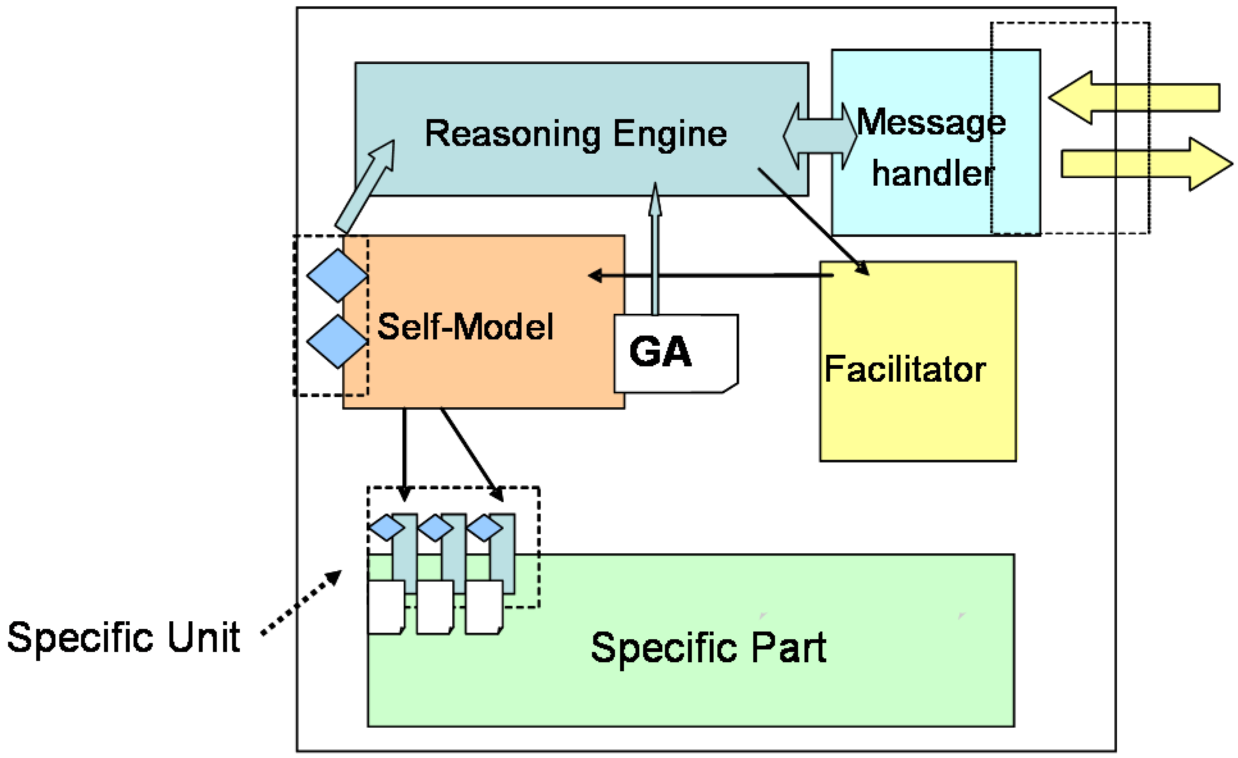

Figure 2: Autonomic Communication Element [25]

model proposed for other functions. The specific feature contains the module that will be invoked or executed to perform a particular job.

Each security feature has a specific interface that specify the Goal Achievable (G_A), i.e. the job the feature is able to do, and a Goal Needed (G_N), i.e. actions or components that are required to provide the job. By applying these concepts in the security framework, we define different types of so-called "security" ACE according to the capabilities and their scope in the network.

Table 2 summarizes the types of *security* Autonomic Communication Elements (ACEs) available in the network and gives example of their scope. The ACEs differentiation is necessary to reduce the complexity of the ACE structure while keeping a specific separation of duties for the components. For instance, we want to define in each ACE a specific feature that is simple and elementary so that we can re-use *security* components when it is required.

The only *security* ACE that implements cryptographic functions is ACE of type B and it is used by other ACEs when security properties should be guaranteed as specified in Section 3. We define a specific class of "security" ACE of type A as they do not have the capabilities to use security services and they cannot work as aggregators and include the self-model of security services or delegate sub-tasks. They are considered passive components in the network.

Advances security services for self-management and self-preservation of the CASCADAS network are demanded to ACEs of type C which are capable of describing in the specific interface as G_A the self-preservation service they provide and in G_N the ACE of type B that are required to protect their communication or to sustain specific tasks. The separation of duties implies the definition of low level granularity for the functions an ACE can perform so that the reutilization and the substitution of a component can be easily done.

Name	Security Capabilities	Security purposes	Example
ACE A	No security capabilities.	It is a passive element that has basic functionalities in the network and local scope. It cannot use other security ACEs during aggregation.	Components of low complexity or low computational abilities, such as a wireless sensor node
ACE B	Minimal security capabilities such as encryption/decryption and means to calculate hash values.	This node can secure messages and provide digital signature services required for authentication. When security is required, it is normally used in a passive way.	PDA or device that runs applications with non sensible data and limited scope.
ACE C	Advanced functionalities such as calculation of reputation and monitoring capabilities	Soft-management of the system and self-preservation capabilities, e.g. prevention of Denial of Service attacks. It makes use of ACE of type B for specific cryptographic funtionalities described in the G_N.	Monitoring or negotiation agents.

Table 2: Different types of security ACEs

In all the three types of ACE, we follow the schema studied in the framework of the definition of the ACE component model and its interactions with the external components. The "security" specific part interacts with the Self-Model, that describes the steps an ACE will execute to achieve its final goal. The adaptation of the basic security functionalities and the transitions between executed intermediate jobs is out of the scope of this deliverable and follows the specification for the *Facilitator* and *Self-Model* respectively defined in WP1.

In the following sections we will describe more in detail the characteristics of each type of ACEs and their conceptual mapping onto the ACE general model. This mapping will also consider the security principles and requirements that are specific for an autonomic communication system as CASCADAS.

7.2 Components without security - ACE type A

These components do not participate into the creation of a security service as aggregators since they do not have sufficient computational capabilities to handle the orchestration of the execution of cryptographic functions. Their role inside the CASCADAS network is limited to act as passive components that can provide simple jobs in a complete unsecure manner.

Due to the possible heterogeneous nature of these components and the different semantic descriptions for the Goal Achievable, ACE A does not carry any description of security actions for the Goal Needed. To accomplish the task described in its specific interface ACE A does not use any ACE of type B and it cannot require the aggregation apart from being passive element that provides execution of a subtask.

The main difference between these components and generic ACEs, that are not part of the security architecture, consists of the limitation that we impose for the description of the Goal Needed.

7.3 Cryptographic functions and services- ACE type B

These components can be considered the first elements that provide security functionalities to the CASCADAS network. Security functionalities are detailed in the specific part which has only the scope of serving other components, aggregators, to secure the communication and respect the security requirements of the job described in the Goal Achievable of the aggregator.

The main parts that characterize this component are:

1. Description of the Goal Achievable
2. Description of the Goal Needed
3. Service Execution and External Data processing

The specification and the mode of communication of the "security" ACE of type B are those specified for any ACE. The reason of this choice comes from the necessity to create security services keeping the simplicity proposed into the ACE model. This enables the use of security services in a complete transparent manner for the ACE aggregator which does not need any dedicated part to communicate or use security functions.

To avoid complication of the model, we have maintained low level granularity for the definition of the capabilities of the "security" ACE. Each simple component has one possible operation, or service implemented, in the specific feature and can provide one specialized service. The specification of the service is semantically described and it is part of the Goal Achievable.

However, security services might not be standalone and require basic functionalities provided by other ACE of type B. For instance a key generator might require the service of a component that has the capabilities to generate cryptographically secure pseudorandom number or of another component that generates random number according to a specific distribution. As we do not include in the specific part multiple security functionalities we stress our concept of low level granularity for a service as the simplest operation an ACE can provide without requiring for its execution other security components or that it will not be provide partially by any other ACE.

This definition of secure component ensures that the network is populated by simple components that can be combined and configured to provide security services adaptively. For instance, malfunctioning components can be replaced easily or the change in the context of the communication might require the adoption of a different functionality, which is not present in the current aggregated component. Different functionalities can be specified *a-priori* as Goal Achievable given specific set of preconditions to activate a particular state of the ACE. Moreover, a monolithic component will be used only when other ACEs specify exactly its Goal Achievable as their Goal Needed limiting the re-use of the network components.

The dynamic composition of this class of security ACE has a key role inside the CASCADAS network as the evolution of the system and its self-preservation relies on the integrity of the feedback coming from system components. Feedbacks are the basis for the system survivability in terms both of context information or reporting opinion on a service/component. Thus, ACE of type B must enable protection of context information in terms of integrity and origin authentication along with confidentiality as it is a sensible data which regulates states changes in an ACE to adapt to the new environment. Self-preservation of the system also relies entirely on the reporting opinions or feedback from nodes to adopt the required strategy or action to avoid mistreatment or attacks from other components. Moreover, context information are available in the aggregate form in general, thus we must avoid poisoning attacks that can render the system instable, less reactive and insecure as we have discussed in Section 3.3. The aggregation of different security capabilities guarantees high flexibility in protecting the information based on the context of the communication and the content to be transferred itself.

7.3.1 Goal Achievable

Main task of a "security" ACE is to provide security services to other ACEs. The message that is used by an ACE to state what kind of job it is able to provide is specified in the Goal Achievable. It has a semantic description of the job. For security purposes, the description of the service is seen within a general framework that includes a set of several ACE and more detailed description that characterize the ACE itself. For instance the Goal Achievable for an ACE might be symmetric cipher function (general description) implementing DES (characterizing function).

We present a set of classes of services that might be part of the ACE B. This list of cryptographic "services" are provided by already exiting approaches that we exploit to construct the basic security framework for CASCADAS. The following description is contextualized into the ACE model and CASCADAS system in general.

Random number generator has the capability to generate a sequence of numbers or bits that appear to not have any correlation among them. A detailed implementation might be represented by generators of numbers according to a specific distribution. This service can be used to generate keys for symmetric key algorithms or to generate nonces to avoid replay attacks or keystreams as input for one-time pad encryption algorithm.

ACE Authentication is required to assess the authenticity of a component. As we deal with devices or part of a device, authentication will be seen as the verification of the ACE identifier or the credential that describe its status. An ACE that implements authentication services can support a mechanism among CHAP [108], PAP [66], Kerberos [55], Public Key authentication and so on.

Authorization is required to avoid improper use of a resource, in this case of the service provided by the ACE. An ACE aggregator to access the functions of an ACE should be authorized and have the right credentials. Authorization can be implemented with policies of access control. An example of authorization service is Kerberos.

Message Authentication is required to authenticate the origin of a message. This cryptographic function is implemented by using a hash function and a secret shared key and produces a message authentication code (MAC). Key hash-based message authentication code (HMAC) might be based on SHA-1 or MD5 [90]. In this case, an ACE implementing HMAC will have as part of the Goal Needed to complete the job a hash function as SHA-1 or MD5.

Integrity guarantees that if a message has been corrupted, the destination can detect data modification. For instance, the implementation of this service can be done by using SHA-1, MD5 or RIPE-MD128/168.

Confidentiality implies that the content of the communication is only disclosed to authorized entities. Confidentiality might be guaranteed by using cryptographic functions to encipher and decipher. The algorithms are divided into families: public key cryptography and symmetric key cryptography (which can be divided in stream and block cipher). Some implementations are DES, 3DES, RC4, Blowfish, AES, RSA, El Gamal, etc..

Non-repudiation is a property to ensure that an action cannot be denied in future. A simple form to implement a non-repudiation service is to use digital signatures. Possible implementations are RSA with hash function, DSA, ElGamal signature scheme, Elliptic curves DSA or ElGamal.

Key agreement is a mechanism to enable two or more parties to agree on a key or to be involved in its determination. A possible implementation of a simple key agreement protocol is Diffie-Helmann.

7.3.2 Goal Needed

The execution of a security service might require functionalities that are not included in the ACE. The Goal Needed is a sort of request, with a semantic description attached, which specifies what kind of functionalities the ACE needs from other ACEs, to achieve its goals. This implies that any ACE should be able, given a G_N, to semantically match it with its Goal Achievable (G_A) in order to properly answer to the received G_N .

For what concerns ACE of type B the Goal Needed will be limited to the description of possible basic cryptographic functions that are required to implement a service. The G_N is in general kept as minimal as possible as this type of ACE is an elementary component and the scope of the G_N message is limited to other ACE of type B.

7.3.3 Execution of security service

The execution of a security service is regulated by the Reason Engine which will keep the state reached by the ACE in the execution of the Self-Model. Mainly, it has to be able to check if a transition may take place in the execution of a service and semantically describe the state reached.

For what concern ACE of type B, the execution is limited to the security service implemented in the specific part. The Reason Engine will invoke the specific security functionality and check whether a transition might occur if the conditions are met.

Security services of type B might require the aggregation with other ACEs of the same type following the protocol G_A-G_N specified in WP1. In this case the specific part will contain the specific features of the aggregated components and their Self-Models.

7.4 Advanced security services - ACE type C

The last level of security components is represented by ACE of type C which will provide advanced security services. Their capability will be to monitor and to self-preserve the CASCADAS network from internal or external attacks.

Internal attacks can be driven by malicious or compromised components that might deviate from the aggregation protocol with the goal to disrupt the network or to take advantage from the services provided by the network. For instance, an ACE might want to use the services of another ACE without contributing to the network; this behaviour is proper of so called *free-riders*.

External attacks can be accomplished by malicious entities who want to disrupt the network and avoid the system to function properly by limiting service provision. An example of external attack is a Denial of Service attack.

The main scope of these ACEs of type C is to self-manage and self-organize the network so that cooperation is exploited also among ACE under different administrative domains.

ACEs of type C use the functions provided by ACEs of type B and other ACEs to build a secure web of trust in an open and heterogeneous network such as the one defined in CASCADAS. ACEs have themselves functionalities to make them able to self-organize and to create a self-defensive operating environment.

In the next Sections, we will present properties (in terms of G_A and G_N) of this type of ACE and then we will detail their use through a critical example of the type of threats that would require such an advanced capability: the distributed denial-or-service, which hide under the cover of valid user traffic from compromised, but authorized and authenticated users and computers. We describe a possible self-protective ACE architecture that would help mitigate the cost of such attacks. The architecture consists of two schemes. The first scheme takes advantage of the expected self-monitoring capabilities of ACEs or relevant knowledge networks to detect attack traffic and allow ACEs make corrective actions. The second extends the previous scheme with prioritisation and throttling to reduce collateral damage to normal traffic.

7.4.1 Goal Achievable

The job of the third level of ACEs consists in self-managing and self-preserving the system. This functionality is not specifically a cryptographic protection of the data or access verification to resources, which are demanded to ACE of type B, but is required to ensure that components follow their duties and the protocol that foster interaction and cooperation among entities.

The G_A-G_N protocol envisioned in WP1 assumes cooperation of ACEs that aggregate or interact upon request. Malicious behaviour of the components will compromise the system as a whole, since aggregated components rely part of their job on corrupted ACE who might change the service provided. For instance, an ACE can insert malicious code such as trojan horse or a virus inside the code representing the service itself.

The set of service that we envision for this class of ACE must detect malicious behaviour and exclude these components from the network so that the system is preserved from both external and internal attacks. Moreover, we include in this class of ACEs also the rational part of a component that should monitor possible resource misuse that some ACEs might perform to take profit from an interaction and at the same time reduce the social utility (or cooperation) inside the system.

7.4.2 Goal Needed

The Goal Needed can be either the service of another security ACE, both of type B and C, or of another generic component. We define the need of ACE of type B for advanced service as cryptographic functions might be required to protect the communication or to avoid reputation of actions of some components.

7.4.3 Security model for Self-Preservation: the case of DoS attack

The definition of the ACE is under way and any security proposal will necessarily depend on the finalized ACE architecture. However, by making certain assumptions, it is possible to suggest a self-defending architecture based on the ideas currently available. The security model that is proposed in this document applies to ACEs responsible for handling communications for applications in a peer to peer-type of network.

We assume that ACEs will aggregate to create communication paths based on a defined needed-goal (G_N) parameter. Convenient aggregations, which could be characterized by a dynamically changing achievable-goal (G_A) value, would be selected. Self-adaptation and self-organization would attempt to maintain the ACE that provide a G_A close to the desired ACE that require its services and publish the G_N. From a network context perspective, such goals will be expressed in terms of a quality-of-service (QoS) metric (e.g. delay, loss, jitter, or other composite metric). We further assume that network metrics will be continuously monitored by ACEs either by implementing the functionally themselves or by relying on another situation-aware network able to monitor and report the QoS of paths. Despite the fact that any ACE could incorporate the functionality required to create a self-defending network, we assume that this functionality will be restricted to a certain class of ACE (henceforth referred to as 'cops'), conveniently deployed to achieve protection. That way, networkwide DDoS protection would exist without excessive overhead.

A complete protection architecture should include the following elements:

- Detection of the existence of an attack. The detection can be either anomaly-based or signature-based, or a hybrid of the two. In anomaly-based detection, the system recognises a deviation from the standard behaviour of its clients, while in the latter it tries to identify the characteristics of known attack types.
- Classification of the incoming packets into valid (normal packets) and invalid (DDoS packets). As in detection, one can choose between anomaly-based and signature-based classification techniques.
- Response. In the most general sense, the protection system either drops the attacking packets or it redirects them into a trap for further evaluation and analysis.

Therefore we start with the postulate that we will consider a generic DDoS defence scheme that is based on the following principles:

- The ACE which is targeted by a DDoS attack (the victim ACE) has the ability to detect or to be informed about the attack, based either on a local or distributed detection scheme. All ACEs upstream of the victim, up to the source (or sources) of the attack, will also be informed of the ongoing threat.
- The victim ACE and the informed ACEs will react by dropping packets which are thought to be part of the attack.
- The attack itself can produce buffer overflows and saturation of network resources such as CPU capacity, due to the inability of the ACEs or routers to handle the resulting heavy packet traffic.
- The detection scheme is always imperfect, so that both false alarms and detection failures are possible. Imperfections are possible both with regard to the detection of the attack as a whole, and the identification of the packets that belong to this attack. Thus, for any packet that flows in the network we need to consider a probability of correct identification as being an attacking packet, and a probability of false alarm, which means that some attacking packets will be missed and some non-attacking packets may be incorrectly dropped.

7.4.3.1 Defence Schemes: Basic Defence Scheme

To provide the necessary defence against DDoS attacks, ACEs will require the ability to trace traffic going both down and upstream. A way this requirement can be achieved is by introducing acknowledgements in end-to-end communications to carry network context information back to the ACEs. When an ACE detects a DDoS attack, it will use the ACKs to ask all intermediate ACEs upstream to drop packets of the incoming flow.

Detection can be achieved by allowing any ACE to determine for itself two parameters governing bandwidth allocation: the maximum that it is able to receive (B_{TOT}), and the maximum that it is willing to allocate to any particular flow that traverses it (B_{Client}). Both are dynamic parameters that may change over time as a function of the conditions at the ACE, and on the identity and G_N (QoS needs) of the flows. They may also vary during the life of a particular flow or connection. This idea can be extended to allowing an ACE to specify different bandwidth restrictions for flows of different QoS classes. When an ACE receives a packet from a flow that it has not already seen before (e.g. with a new source-destination pair, accompanied with a new G_N), it will send a specific Flow-ACK packet back to the source along the reverse path, and inform the source of its B_{Client} allocation. This may occur periodically for each ongoing flow. An ACE will monitor all of the flows that traverse it, and drop some or all of the packets of any flow that exceeds this allocation. When the allocation is exceeded, the ACE informs (using ACKs) upstream ACEs that packets of this flow should be dropped. Other possible actions could include diverting the flow into a "honey-pot", or into a special overlay network used for protection, or it may simply alert a network administrator.

7.4.3.2 Defence Schemes: DDoS Protection based on Prioritisation and Throttling

The performance of the defence scheme based on dropping DDoS packets previously described depends heavily on the accuracy of the detection and classification methods available. Inaccurate methods would easily lead to false alarms on normal packet flows causing a degradation in communications.

We suggest traffic prioritisation and throttling as the basis of the response mechanism instead of a simple dropping scheme. If an ACE (either a recipient or a transit ACE) receives packets towards a destination: (1) at a rate higher than the current rate threshold (packets/sec or bytes/sec) and (2) with a rate increase higher than the current increase threshold (packets/sec^2 or bytes/sec^2), then it announces the existence of an ongoing DDoS attack and sends this information to all upstream ACEs and to the victim. From then on, the protection mechanism is set into motion along the informed path. In case of disagreement, it is the alleged victim ACE's responsibility to inform those ACEs that there was a false alarm and that they should return to normal operation.

When there is a detected attack in progress, each informed ACE contributes to the defence by examining every incoming packet for deviations from the normal behaviour. Packets undergo a collection of anomaly-based validity tests which may differ for each type of traffic (see Section 1.2). ACEs which have a role in the defence will prioritise traffic with priority levels which are related to the tests. Each time a packet fails a validity test, its priority level may drop accordingly. Additionally, the upstream routers are instructed to throttle down their traffic directed towards the victim ACE to a level which it handles. This two-fold protection framework ensures that packets with higher probability of being both valid and harmless are offered preferential service. Packets which have been marginally classified as invalid may now receive service if there is available bandwidth so as to minimise the collateral damage inflicted by false detection. Packets which have been identified as being harmful are either delayed by being assigned low priority, or dropped. Various simplifications of this scheme, based for instance on grouping all traffic that has been identified as being invalid, can also be considered.

8 Mock up of ACE interactions

In this section use the *behavioural pervasive advertisement* scenario to illustrate some of the functionality of the proposed security architecture.

The behavioural pervasive advertisement (BPA) scenario would make an attractive target for security attacks. In order effectively function (i.e. autonomously provide advertising tailored to the current audience), such a system would need to anonymously gather information on shopping habits, personal interests, and movements for single individuals or large groups, depending on how targeted the advertising is. Users of the mobile devices, which will play a large part in gathering such information, are likely to be hostile towards such a system. They would likely opt-out of providing the data unless strict anonymity and security guidelines are in force, guaranteeing that any data gathered can, and will, not be used for more than the selection of advertising (and perhaps some statistical analysis). Such information could however be of great interest to unauthorised third-parties, and must therefore remain secure and anonymous.

The dynamic advertising may itself also become a target. A group or individual may wish utilise flaws in the system to advertise their products or services without authorisation. They may wish to vandalise authorised advertisings, in a disparaging manner, or simply target certain advertisers by obstructing their ability to display their adverts. An authorised yet disruptive user could also undermine the effectiveness of the system by reporting false information in order to influencing the advertisements shown.

We assume that the BPA system will make use of dynamically forming autonomous ad-hoc networks between personal devices with wireless networking functionalities, which opportunistically connect to static network infrastructures and other ad-hoc networks. For this example we will furthermore assume that the advertising can be either in a public place with a potentially large audience (for example, the central area of an indoor shopping complex) or targeted at an individual (for example, a clothes store changing room).

8.1 Securing Inter-ACE Communication

Securing communications between devices in the behavioural pervasive advertising scenario involves several stages. They are outlined in Figure 3 that shows the G_N-G_A message exchanges for the BPA scenario, and Figure 4 that depicts the interactions between ACEs once the connections have been made.

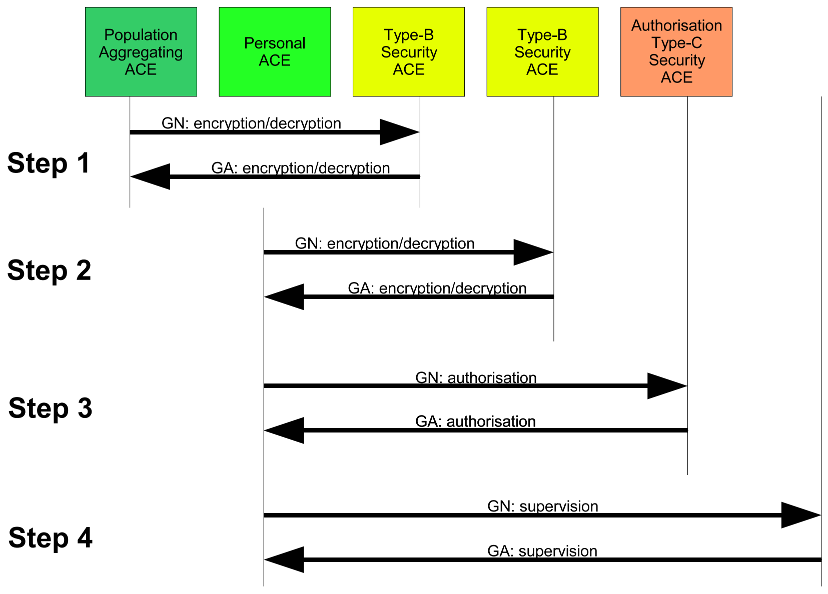
Population Aggregating ACE
Personal ACE
Type-B Security ACE
Type-B Security ACE
Authorisation Type-C Security ACE
Step 1
GN: encryption/decryption
GA: encryption/decryption
Step 2
GN: encryption/decryption
GA: encryption/decryption
Step 3
GN: authorisation
GA: authorisation
Step 4
GN: supervision
GA: supervision

Figure 3: ACE G_N-G_A message exchanges

- First a secure connection must be established between the communicating devices. This could be a direct connection between the BPA display and the mobile device of an individual in the changing room example, and via a multi-hop network in the shopping complex example. The *population aggregating ACE* of the BPA display will advertise a *goal needed* of message encryption/decryption. A type-B security ACE from the BPA display with this *goal achievable* will respond and, aggregate the first ACE (*Step 1* in Figure 3).
- The same will occur between the data gathering *personal ACEs* on the mobile device of an individual and another type-B security ACE (*Step 2* in Figure 3).
- After a public key exchange, secured communication can take place between the *population aggregating ACE* and the *personal ACEs*. This ensures that the data cannot be read by other personal ACEs or other devices forming the ad-hoc network.
- Before sharing personal, potentially sensitive data with a population aggregating ACE, the personal ACEs will establish its authenticity. They advertise a G_N of *authorisation* in order to establish if the aggregating ACE has the correct credentials. A type-C security ACE with this G_A, which may require a connection to a trusted third party authorisation server which the personal ACE trusts,

can perform this task. If authorisation is granted, the personal data can be anonymously shared. If no ACE with such functionality is available, no data is shared (*Step 3* in Figure 3).

- While operating, the population aggregating ACE must be supervised by a third-party authority (supervising type-C security ACE) in order to ensure it has not been compromised, and that it is honouring its confidentiality commitments. If a security breach is observed, personal ACEs can be informed (through the third party authorisation server) to no longer share data (*Step 4* in Figure 3).

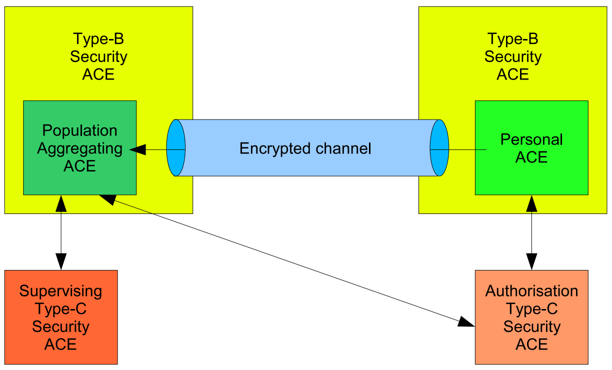
Type-B Security ACE
Population Aggregating ACE
Encrypted channel
Type-B Security ACE
Personal ACE
Supervising Type-C Security ACE
Authorisation Type-C Security ACE

Figure 4: ACE Interactions

8.2 Securing Network Infrastructure

Type-C security ACEs provide a more advanced set of security capabilities, and are to be a large part of future research. An example of a type-C capability is the ability to self protect against denial of service attacks. The ad-hoc network being used to aggregate the personal ACEs may become the target of such a denial of service attack, which could be used to compromise the operation of the BPA system, or in order to enable or mask other security breaches. Much of this work is not finalised, but the network is expected to self protect against DoS attacks in the following manner:

- Secure connections between communicating nodes are established as above. In addition a dynamic service level agreement, stating bandwidth allowances, may be established based on the capabilities of the network and number of other flows communicating with the host.
- The ACEs providing network communications will then continuously monitor network quality of service metrics. This can be done using message acknowledgements. The QoS data is stored in ACE knowledge networks, as will be defined by the research conducted by WP5.
- Type-C ACE "cops" will monitor these knowledge networks for anomalous traffic conditions, or for broken service level agreements, which may indicate a DoS attack. If a break in the SLA is detected by the host ACE, it can inform the ACE cops.
- When an attack is suspected, the ACE cops inform the upstream network infrastructure ACEs to drop traffic from the offending flows, or to prioritise and throttle flows as is discussed in the final part of section 7. (How flows are to be identified is highly dependant on how individual ACEs are to be identified, and the trust that is assigned to them to identify themselves correctly. ACE identifiers are future work, and are closely tied to the advanced capabilities of the security architecture.)

- The cops continue to monitor the knowledge networks for changes in conditions indicating that normal use of the network can be resumed.
- As specifically relates to the PBA scenario, once an attack is detected, personal ACEs are informed that the network is potentially compromised, and to not share data.

8.3 Conclusions from the Mock-up

In the personal behavioural advertising scenario, the primary concern from a subject's point of view is the confidentiality of his or her personal information. For the advertisers it is the effectiveness of the network in supporting the advertising system. The proposed CASCADAS security architecture, once developed fully, should provide for both of these goals.

9 Conclusion

In this Deliverable we have defined a framework to provide traditional security and soft security services to the CASCADAS network. Traditional security mechanisms and algorithms have been explored in the context of a situated autonomic communication system, thus they extend and in some cases re-invent the way classical cryptographic schemes have been defined in the literature.

The vast area of mobile code security, which is well suited to the autonomic communication element (ACE) abstraction, has been explored, pinning down the culprit for vulnerabilities that might arise in the CASCADAS network.

Soft security mechanisms reflect the vision for the CASCADAS project that extends the boolean perception of security (a system is secure or compromised) to incorporate a gray-shaded region wherein attacks to a system have not the goal of jeopardizing its correct functioning but rather are motivated by self-interest or even anti-social behaviour. The area of game theory and mechanism design provide a solid yet not widely explored ground to study the problem of selfishness and lack of cooperation in order to study the effects of such behaviour on the CASCADAS network and to come up with adequate countermeasures.

Particular attention has been devoted to the integration of the security components and architecture envisioned in WP4 of the CASCADAS project with the services addressed by other work packages of the project. We briefly summarize our findings hereafter:

- *Integration with WP2: Pervasive Supervision*. The core cryptographic concepts presented in this Deliverable underpin the monitoring activities carried out by the CASCADAS by enabling not only the basic security requirements for secure information sharing and dissemination in the CASCADAS network, but also supporting advanced features such as secure information aggregation, and more generically, secure in-network processing. To an even larger extent, in this Deliverable we covered issues related to mobile code security, that might be well suited for a realistic and secure deployment of a CASCADAS service network.
- *Integration with WP3: Self Aggregation*. The autonomic activity of aggregating ACEs component into clusters (or the reverse operation of regrouping ACEs so as to achieve better service diversity) has also been integrated in the CASCADAS vision of security. In this Deliverable we pointed out the main research directions into which CASCADAS may evolve towards the definition of realistic attack models, both being based on traditional models and game theoretical models. A tight integration of the game theoretical modelling activity and traditional security mechanisms (but in a distributed and autonomic context) is clearly discussed in this Deliverable.

- *Integration with WP5: Knowledge Networks*. Similarly to the requirements of the monitoring capabilities of the CASCADAS architecture (addressed in WP2), the diffusion of knowledge and in general information calls for both standard security mechanisms and advanced features that we discussed in this Deliverable taking several perspectives. Moreover, the decision making process that characterizes a system able to self-protect against known and unknown attacks is based on critical system state knowledge, that we characterize in CASCADAS under the term of context awareness. One of the most prominent is the use of knowledge so as to react to unexpected or undesired system states deriving for example by (distributed) Denial of Service attacks, that are addressed in depth in this Deliverable

The security architecture resulting from our research work is flexible and accounts for the primitive definition of an ACE as defined by WP1, the basic communication element of the CASCADAS architecture. The CASCADAS security architecture provides basic security features using traditional cryptographic protocols, however, it greatly enhances the way cryptographic primitives are generated and exchanged in the system. Furthermore, with the aim of providing a comprehensive set of security services for the advanced components that constitute the CASCADAS architecture, the need for specialized security algorithms and protocols has been addressed emphasizing the innovation they bring to the security architecture and to the CASCADAS project.

10 Bibliography

[1] Karl Aberer and Zoran Despotovic. Managing trust in a peer-2-peer information system. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, pages 310–317, Atlanta, Georgia, USA, 2001. ACM Press.

[2] WP3 Self-Organised Component Aggregation and Emergent System Properties. Aggregation algorithms, overlay dynamics and implications for self-organised distributed systems. Deliverable D3.1, CASCADAS Consortium, January 2007.

[3] M. Alfalayleh and L. Brankovic. An overview of security issues and techniques in mobile agents. In *Proceedings of the Eighth IFIP TC-6 TC-11 Conference on Communications and Multimedia Security (CMS2004)*, 2004.

[4] D. Andersen. Mayday: Distributed DoS filtering for internet services. *ACM SIGCOMM Computer Communication Review*, 34(1):39–44, 2004.

[5] R. Anderson, H. Chan, and A. Perrig. Key infection: smart trust for smart dust. In *Proc. of IEEE ICNP*, 2004.

[6] Elisa Bertino, Elena Ferrari, and Anna Squicciarini. Trust-X: A Peer-to-Peer Framework for Trust Establishment. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):827–842, 2004.

[7] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote Trust-Management System Version 2. RFC 2704, IETF, September 1999.

[8] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The role of trust management in distributed systems security. In *Secure Internet programming: security issues for mobile and distributed objects*, volume 1603 of *Lecture Notes in Computer Science*, pages 185–210. Springer-Verlag, London, UK, 1999.

[9] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized Trust Management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy (SP '96)*, pages 164–173, Oakland, CA, USA, May 6-8 1996. IEEE Computer Society.

[10] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *In Lecture Notes in Computer Science*, 2139, 2001.

[11] S. Capkun, L. Buttyan, and J. Hubaux. Self-organized public-key management for mobile ad hoc networks. In *Proceedings of ACM International Workshop on Wireless Security (WISE)*, 2002.

[12] CERT Coordination Center. Denial of service attack via ping, December 1997. CERT Advisory CA-1996-26. http://www.cert.org/advisories/CA-1996-26.html.

[13] CERT Coordination Center. MS-SQL server worm, January 2003. CERT Advisory CA-2003-04. http://www.cert.org/advisories/CA-2003-04.html.

[14] K. Cetnarowicz and G. Rojek. Behavior based detection of unfavourable resource. In *Lecture Notes in Computer Science*, volume 3038, pages 607–614, 2004.

[15] A. P. F. Chan, W. W. Y. Ng, D. S. Yeung, and E. C. C. Tsang. Multiple classifier system with feature grouping for intrusion detection: Mutual information approach. In *Lecture Notes in Artificial Intelligence*, volume 3683, pages 141–148, 2005.

[16] J.C. Cha and J.H. Cheon. An identity-based signature from gap diffie-hellman groups. In *Proceedgins of the International Workshop on Practice and Theory in Public Key Cryptography, PKC*. LNCS, 2003.

[17] I. A. Cheong, Y. M. Kim, M. S. Kim, and B. N. Noh. The causality analysis of protocol measures for detection of attacks based on network. In *Lecture Notes in Computer Science*, volume 3090, pages 962–972, 2004.

[18] Yang-Hua Chu, Joan Feigenbaum, Brian LaMacchia, Paul Resnick, and Martin Strauss. REFEREE: trust management for web applications. *World Wide Web J.*, 2(3):127–139, 1997.

[19] Peter Clark and Tim Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261–283, March 1989.

[20] C. Cocks. An identity based encryption scheme based on quadratic residues. *Lecture Notes in Computer Science*, 2260, 2001.

[21] CASCADAS Consortium. Annex I - "Description of Work". Technical Annex, October 2005.

[22] Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, Pierangela Samarati, and Fabio Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 207–216, Washington, DC, USA, 2002. ACM Press.

[23] X. Ding and G. Tsudik. Simple identity-based cryptography with mediated RSA. In *CTRSA: CT-RSA, The Cryptographers' Track at RSA Conference, LNCS*, 2003.

[24] John R. Douceur. The Sybil Attack. In *IPTPS '01: First International Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK, 2002. Springer-Verlag.

[25] WP1 Autonomic Communication Elements. Report on state-of-art, requirements and ACE model. Deliverable D1.1, CASCADAS Consortium, January 2007.

[26] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI Certificate Theory. RFC 2693, IETF, September 1999.

[27] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[28] William M. Farmer, Joshua D. Guttman, and Vipin Swarup. Security for mobile agents: Authentication and state appraisal. In *ESORICS ’96: Proceedings of the 4th European Symposium on Research in Computer Security*, pages 118–130. Springer-Verlag, 1996.

[29] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred. Statistical approaches to DDoS attack detection and response. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX’03)*, 2003.

[30] P. Ferguson and D. Senie. Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing. Technical Report RFC 2267, IETF, January 1998.

[31] Lee Garber. Denial-of-service attacks rip the internet. *Computer*, 33:12–17, April 2000.

[32] A. Garg, R. Battiti, and R. Cascella. Reputation management: Experiments on the robustness of ROCQ. In *Proceedings of the 7th International Symposium on Autonomous Decentralized Systems (First International Workshop on Autonomic Communication for Evolvable Next Generation Networks)*, pages 725–730, Chengdu, China, April 2005.

[33] A. Garg, R. Battiti, and G. Costanzi. Dynamic self-management of autonomic systems: The reputation, quality and credibility (RQC) scheme. In *The 1st IFIP TC6 WG6.6 International Workshop on Autonomic Communication (WAC 2004)*, October 2004.

[34] D. Gavrilis and E. Dermatas. Real-time detection of distributed denial-of-service attacks using RBF networks and statistical features. *Computer Networks*, 48:235–245, 2005.

[35] D. Gavrilis, I. Tsoulos, and E. Dermatas. Feature selection for robust detection of distributed denial-of-service attacks using genetic algorithms. In *Lecture Notes in Artificial Intelligence*, volume 3025, pages 276–281, 2004.

[36] G. Giacinto, F. Roli, and L. Didaci. Fusion of multiple classifiers for intrusion detection in computer networks. *Pattern Recognition Letters*, 24(12):1795–1803, 2003.

[37] Tyrone Grandisonm and Morris Sloman. A survey of trust in internet application. *IEEE Communications Surveys & Tutorials*, 3(4), 2000.

[38] Michael S. Greenberg, Jennifer C. Byington, and David G. Harper. Mobile agents and security. *IEEE Communications Magazine*, 36(7), 1998.

[39] PKIX Working Group. Public-key infrastructure (x.509) (pkix).

[40] R. Gu, P. Yan, T. Zou, and C. Guo. An automatic and generic early-bird system for internet backbone based on traffic anomaly detection. In *Lecture Notes in Computer Science*, volume 3420, pages 740–748, 2005.

[41] F. Hess. Efficient identity based signature schemes based on pairings. In *Selected Areas in Cryptography, SAC*, pages 310–324. Springer-Verlag, February 2002.

[42] H. T. He, X. N. Luo, and B. L. Liu. Detecting anomalous network traffic with combined fuzzy-based approaches. In *Lecture Notes in Computer Science*, volume 3645, pages 433–442, 2005.

[43] Fritz Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. In *Mobile Agents and Security*, pages 92–113, London, UK, 1998. Springer-Verlag.

[44] A. Hussain, J. Heidemann, and C. Papadopoulos. Distinguishing between single and multi-source attacks using signal processing. *IEEE Computer Networks*, 46:479–503, 2004.

[45] ITU-T. The directory - public-key and attribute certificate frameworks. Technical Report Recommendation X.509, ITU-T, Geneva, Switzerland, 2000.

[46] R. Jalili, F. Imani-Mehr, M. Amini, and H.-R. Shahriari. Detection of distributed denial of service attacks using statistical pre-processor and unsupervised neural networks. In *Lecture Notes in Computer Science*, volume 3439, pages 192–203, 2005.

[47] Wayne A. Jansen. Countermeasures for mobile agent security. *Computer Communications*, 23:1667–1676, 2000.

[48] W. Jansen and T. Karygiannis. Nist special publication 800-19: Mobile agent security. Technical report, National Institute of Standards and Technology, Computer Security Division, August 1999.

[49] Shudong Jin and Azer Bestavros. Sources and Characteristics of Web Temporal Locality. In *Proceedings of Mascots'2000: The IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, San Fransisco, CA, August 2000.

[50] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 2007. In Press, Corrected Proof, Available online 5 July 2005.

[51] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 640–651, Budapest, Hungary, 2003. ACM Press.

[52] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure overlay services. In *Proceedings of ACM SIGCOMM*, pages 61–72, Pittsburgh, PA, USA, 2002.

[53] S. M. Khattab, C. Sangpachatanaruk, R. Melhern, D. Mosse, and T. Znati. Proactive server roaming for mitigating denial-of-service attacks. In *Proceedings of International Conference on Information Technology Research and Education*, pages 289–290, August 2003.

[54] M. Kim, H. Na, K. Chae, H. Bang, and J. Na. A combined data mining approach for DDoS attack detection. In *Lecture Notes in Computer Science*, volume 3090, pages 943–950, 2004.

[55] J. Kohl and C. Neuman. The Kerberos Network Authentication Service (V5). RFC 1510, IETF, September 1993.

[56] Hristo Koshutanski and Fabio Massacci. E Pluribus Unum: Deduction, Abduction and Induction, the Reasoning Services for Access Control in Autonomic Communication. In *First International IFIP Workshop on Autonomic Communcation (WAC 2004)*, volume 3457 of *Lecture Notes in Computer Science*, pages 179–190, Berlin, Germany, October 18-19 2004.

[57] A. B. Kulkarni, S. F. Bush, and S. C. Evans. Detecting distributed denial-of-service attacks using kolmogorov complexity metrics. Technical report, General Electric Global Research, 2002.

[58] D. Lange. Mobile objects and mobile agents: The future of distributed computing? In *The European Conference on Object Oriented Programming (ECOOP 98)*, Brussels, Belgium, July 1998.

[59] Nikolaos Laoutaris, Georgios Smaragdakis, Azer Bestavros, and Ioannis Stavrakakis. Mistreatment in distributed caching groups: Causes and implications. In *Proceedings of IEEE INFOCOM'06*, Barcelona, Spain, April 2006.

[60] Nikolaos Laoutaris, Orestis Telelis, Vassilios Zissimopoulos, and Ioannis Stavrakakis. Distributed selfish replication. *IEEE Transactions on Parallel and Distributed Systems*, 17(12):1401–1413, December 2006.

[61] P. Lee. Proof-carrying code. Retrieved December 28, 2003, from Web site: URL: http://www-2.cs.cmu.edu/petel/papers/pcc/pcc.html.

[62] Seungjoon Lee, Rob Sherwood, and Bobby Bhattacharjee. Cooperative peer groups in NICE. In *IEEE INFOCOM 2003*, volume 2, pages 1272–1282, San Francisco, CA, USA, April 2003.

[63] S. Y. Lee, Y. S. Kim, B. H. Lee, and S. H. Kang C. H. Youn. A probe detection model using the analysis of the fuzzy cognitive maps. In *Lecture Notes in Computer Science*, volume 3480, pages 320–328, 2005.

[64] M. Li, C.-H. Chi, and D. Long. Fractional gaussian noise: A tool for characterizing traffic for detection purpose. In *Lecture Notes in Computer Science*, volume 3309, pages 94–103, 2004.

[65] M. Li. An approach to reliably identifying signs of DDoS flood attacks based on LRD traffic pattern recognition. *Computers and Security*, 23(7):549–558, 2004.

[66] B. Lloyd and W. Simpson. PPP Authentication Protocols. RFC 1334, IETF, October 1992.

[67] Thanasis Loukopoulos, Petros Lampsas, and Ishfaq Ahmad. Continuous replica placement schemes in distributed systems. In *Proceedings of the 19th ACM International Conference on Supercomputing (ACM ICS)*, Boston, MA, June 2005.

[68] H. Luo and S. Lu. Ubiquitous and robust authentication services for ad hoc wireless networks. Technical Report TR-200030, Dept. of Computer Science, UCLA, 2000.

[69] Siddharth Maini. A Survey Study on Reputation-Based Trust Management in P2P Networks. Technical report.

[70] D. Mankins, R. Krishnan, C. Boyd, J. Zao, and M. Frentz. Mitigating distributed denial of service attacks with dynamic resource pricing. In *Proceedings of Computer Security Applications Conference*, pages 411–421, December 2001.

[71] Sergio Marti and Hector Garcia-Molina. Taxonomy of trust: categorizing p2p reputation systems. *Comput. Networks*, 50(4):472–484, 2006.

[72] Gary McGraw and Edward Felten. *Securing Java: Getting Down to Business with Mobile Code*. John Wiley and Sons, New York, NY, USA; London, UK; Sydney, Australia, 1998.

[73] Loubna Mekouar, Youssef Iraqi, and Raouf Boutaba. Peer-to-peer’s most wanted: malicious peers. *Comput. Networks*, 50(4):545–562, 2006.

[74] Mostafa Ammar Meng Guo. Scalable live video streaming to cooperative clients using time shifting and video patching. In *Proceedings of IEEE INFOCOM’04*, Hong Kong, March 2004.

[75] J. Mirkovic, P. Reiher, and M. Robinson. Forming alliance for DDoS defense. In *New Security Paradigms Workshop*, Centro Stefano Francini, Ascona, Switzerland, August 2003.

[76] J. Mirkovic and P. Reiher. D-WARD: A source-end defense against flooding denial-of-service attacks. *IEEE Transactions on Dependable and Secure Computing*, 2(3):216–232, July 2005.

[77] W. G. Morein, A. Stavrou, D. L. Cook, A. D. Keromytis, V. Mishra, and D. Rubenstein. Using graphic turing tests to counter automated DDoS attacks against web servers. In *Proc. 10th ACM Int'l Conference on Computer and Communications Security (CCS '03)*, pages 8–19, Washington DC, USA, October 2003.

[78] G. Mori and J. Malik. Recognizing objects in adversarial clutter - breaking a visual CAPTCHA. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2003 (CVPR '03)*, volume 1, pages 134–141, Madison, WI, USA, June 2003.

[79] R. T. Morris. A weakness in the 4.2BSD Unix TCP/IP software. Technical Report Computer Science #117, AT&T Bell Labs, February 1985.

[80] S. Mukkamala, A. H. Sung, and A. Abraham. Intrusion detection using an ensemble of intelligent paradigms. *Journal of Network and Computer Applications*, 28:167–182, 2005.

[81] S. Mukkamala and A. H. Sung. Computational intelligent techniques for detecting denial of service attacks. In *Lecture Notes in Artificial Intelligence*, volume 3029, pages 616–624, 2004.

[82] George C. Necula and Peter Lee. Safe kernel extensions without run-time checking. In *OSDI '96: Proceedings of the second USENIX symposium on Operating systems design and implementation*, pages 229–243, Seattle, Washington, United States, 1996. ACM Press.

[83] S. Noh, C. Lee, K. Choi, and G. Jung. Detecting distributed denial of service (DDoS) attacks through inductive learning. In *Lecture Notes in Computer Science*, volume 2690, pages 286–295, 2003.

[84] Joann J. Ordille. When agents roam, who can you trust? In *Proceedings., First Annual Conference on Emerging Technologies and Applications in Communications*, pages 188–191, Portlan, Oregon, USA, 7-10 May 1996.

[85] Christos Papadimitriou. Algorithms, games, and the internet. In *Proceedings of ACM STOC '01*, pages 749–753, Hersonissos, Greece, 2001.

[86] K.G. Paterson. ID-based signatures from pairings on elliptic curves. Report 2002/004, Cryptology ePrint Archive, January 2002.

[87] V. Paxson. An analysis of using reflectors for distributed denial-of-service attacks. *ACM Computer Communications Review*, 31(3), July 2001.

[88] T. Peng, C. Leckie, and K. Ramamohanarao. Detecting distributed denial of service attacks by sharing distributed belief. In *Lecture Notes in Computer Science*, volume 2727, pages 214–225, 2003.

[89] Stefan Podlipnig and Laszlo Böszörmenyi. A survey of web cache replacement strategies. *ACM Computing Surveys*, 35(4):374–398, 2003.

[90] Bart Preneel. Cryptographic hash functions. *European Transactions on Telecommunication and related Tehcnologies*, 5(4):431–448, 1994.

[91] Josep M. Pujol, Ramon Sanguesa, and Jordi Delgado. Extracting reputation in multi agent systems by means of social network topology. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 467–474, Bologna, Italy, 2002. ACM Press.

[92] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[93] Sanjay Rao. Establishing the viability of end system multicast using a systems approach to protocol design. *Phd Thesis, Technical Report CMU-CS-04-168, Carnegie Mellon University*, October 2004.

[94] Lars Rasmusson and Sverker Jansson. Simulated social control for secure internet commerce. In *NSPW ’96: Proceedings of the 1996 workshop on New security paradigms*, pages 18–25, Lake Arrowhead, California, United States, 1996. ACM Press.

[95] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. Reputation systems: Facilitating trust in internet applications. *Communications of the ACM*, 43(12):45–48, 2000.

[96] James Riordan and Bruce Schneier. Environmental key generation towards clueless agents. In *Mobile Agents and Security*, pages 15–24, London, UK, 1998. Springer-Verlag.

[97] R. Rivest, A. Shamir, and L. Adelman. A method for obtaining digital signatures and public key cryptosystems. *Communications of ACM*, 21:120–126, 1978.

[98] Martin Roesch. Snort - lightweight intrusion detection for networks. *Proceedings of LISA ’99: 13th Systems Administration Conference*, pages 229–238, November 1999.

[99] Keith W. Ross. Hash-routing for collections of shared web caches. *IEEE Network*, 11(6), November 1997.

[100] Aviel D. Rubin and Jr. Daniel E. Geer. Mobile code security. *IEEE Internet Computing*, 2(6):30–34, 1998.

[101] Jordi Sabater and Carles Sierra. Reputation and social network analysis in multi-agent systems. In *AAMAS ’02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 475–482, Bologna, Italy, 2002. ACM Press.

[102] Tomas Sander and Christian F. Tschudin. Protecting mobile agents against malicious hosts. In *Mobile Agents and Security*, pages 44–60, London, UK, 1998. Springer-Verlag.

[103] Michael Schillo, Petra Funk, and Michael Rovatsos. Using trust for detecting deceitful agents in artificial societites. *Applied Artificial Intelligence Journal, Special Issue on Trust, Deception and Fraud in Agent Societies*, 14, 2000.

[104] H. S. Seo and T. H. Cho. Modeling and simulation for detecting a distributed denial of service attack. In *Lecture Notes in Artificial Intelligence*, volume 2557, pages 179–190, 2002.

[105] A. Shamir. Identity based cryptosystems and signature schemes. In *Advances in Cryptography*. ePress, 1984.

[106] M. S. Shin, E. H. Kim, and K. H. Ryu. False alarm classification model for network-based intrusion detection system. In *Lecture Notes in Computer Science*, volume 3177, pages 259–265, 2004.

[107] C. Siaterlis and B. Maglaris. Towards Multisensor Data Fusion for DoS Detection. In *2004 ACM Symposium on Applied Computing*, pages 439–446, 2004.

[108] W. Simpson. PPP Challenge Handshake Authentication Protocol (CHAP). RFC 1994, IETF, August 1996.

[109] Christopher Small and Margo I. Seltzer. A comparison of OS extension technologies. In *USENIX Annual Technical Conference*, pages 41–54, 1996.

[110] Georgios Smaragdakis, Nikolaos Laoutaris, Ibrahim Matta, Azer Bestavros, and Ioannis Stavrakakis. A feedback control approach to mitigating mistreatment in distributed caching groups. In *Proceedings of IFIP Networking 2006*, Coimbra, Portugal, May 2006.

[111] A. H. Sung and S. Mukkamala. The feature selection and intrusion detection problems. In *Lecture Notes in Computer Science*, volume 3321, pages 468–482, 2004.

[112] Girish Suryanarayana and Richard N. Taylor. A survey of trust management and resource discovery technologies in peer-to-peer applications. Technical Report UCI-ISR-04-6, Institute for Software Research (ISR), University of California, Irvine, July 2004.

[113] Hock Kim Tan and Luc Moreau. Extending execution tracing for mobile code security. In *Proceedings of Second International Workshop on Security of Mobile MultiAgent Systems (SEMAS'2002)*, pages 51–59, Bologna, Italy, June 2002.

[114] R. Thomas, B. Mark, T. Johnson, and J. Croall. Netbouncer: client-legitimacy-based high-performance DDoS filtering. In *Proc. DARPA Information Survivability Conference and Exposition*, volume 1, pages 14–25, April 2003.

[115] UK Security Online. Denial of service attack threat analysed, 2002.

[116] C. Vassilakis, N. Laoutaris, and I. Stavrakakis. On the benefits of synchronized playout in peer to peer streaming. In *CoNEXT'06 student Workshop, poster presentation*, Lisbon, Portugal, December 2006.

[117] G. Vigna. Protecting Mobile Agents through Tracing. In *Proceedings of the Third International Workshop on Mobile Object Systems, in conjunction with the 11^{th} European Conference on Object-Oriented Programming (ECOOP '97)*, Jyväskylä, Finland, June 1997. Online Proceedings.

[118] P. A. Chou V. N. Padmanabhan, H. J. Wang. Resilient peer-to-peer streaming. In *Proceedings of ICNP'03*, November 2003.

[119] Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham. Efficient software-based fault isolation. *ACM SIGOPS Operating Systems Review*, 27(5):203–216, December 1993.

[120] Stephen Weeks. Understanding Trust Management Systems. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy (SP '01)*, pages 94–105, Oakland, California, USA, May 14-17 2001. IEEE Computer Society.

[121] William H. Winsborough, Kent E. Seamons, and Vicki E. Jones. Automated Trust Negotiation. In *Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX '00)*, volume 1, pages 88–102. IEEE Press, January 25-27 2000.

[122] Marianne Winslett, Ting Yu, Kent E. Seamons, Adam Hess, Jared Jacobson, Ryan Jarvis, Bryan Smith, and Lina Yu. Negotiating trust on the web. *IEEE Internet Computing*, 6(6):30–37, 2002.

[123] Alec Wolman, M. Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin, and Henry M. Levy. On the scale and performance of cooperative Web proxy caching. *SIGOPS Oper. Syst. Rev.*, 33(5):16–31, 1999.

[124] Gregory Wroblewski. *General Method of Program Code Obfuscation*. PhD dissertation, Fanstord University, Wroclaw University of Technology, Institute of Engineering Cybernetics, 2002.

[125] Y. Xiang, Y. Lin, W. L. Lei, and S. J. Huang. Detecting DDoS attack based on network self-similarity. In *IEE Proceedings in Communication*, volume 151, pages 292–295, 2004.

[126] R. R. Yager, J. Kacprzyk, and M. Fedrizzi. *Advances in the Dempster-Shafer theory of evidence*. John Wiley & Sons, 1994.

[127] Liangzhong Yin and Guohong Cao. Supporting cooperative caching in ad hoc networks. In *INFOCOM*, 2004.

[128] Fabian E. Bustamante Yi Qiao. Elders know best - handling churn in less structured p2p systems. In *Proceedings of Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005)*, Konstanz, Germany, August 2005.

[129] Adam Young and Moti Yung. Sliding encryption: A cryptographic tool for mobile agents. In *FSE '97: Proceedings of the 4th International Workshop on Fast Software Encryption*, pages 230–241, London, UK, 1997. Springer-Verlag.

[130] Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transaction in Information and System Security*, 6(1):1–42, 2003.

[131] Ting Yu and Marianne Winslett. A Unified Scheme for Resource Protection in Automated Trust Negotiation. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy (SP '03)*, pages 110–122, Oakland, California, USA, May 11-14 2003. IEEE Computer Society.

[132] S. Zhu, S. Xu, S. Setia, and S. Jajodia. Establishing pair-wise keys for secure communication in ad hoc networks: A probabilistic approach. Technical Rerport ISE-TR-03-01, George Mason University, March 2003.

[133] Phil Zimmermann. *The Official* PGP *User´s Guide*. M.I.T. Press, 1995.