JACET Journal of Advances in Computer Engineering and Technology, 3(2) 2017

P2P Network Trust Management Survey

Seyed Hossein Ahmadpanah[1], Rozita Jamili Oskouei[2], Abdullah Jafari Chashmi[3]

Received (2017-01-06)
Accepted (2017-04-10)

Abstract — **Peer-to-peer applications (P2P) are no longer limited to home users, and start being accepted in academic and corporate environments. While file sharing and instant messaging applications are the most traditional examples, they are no longer the only ones benefiting from the potential advantages of P2P networks. For example, network file storage, data transmission, distributed computing, and collaboration systems have also taken advantage of such networks. In this paper, we will present a summary of the main safety aspects to be considered in P2P networks, highlighting its importance for the development of P2P applications and systems on the Internet and deployment of enterprise applications with more critical needs in terms of security .**

Index Terms — **Peer to Peer Network, File Sharing, Napster, Bit Torrent, Trust.**

1- Department of Computer and Information Technology, Mahdishahr Branch, Islamic Azad University, Mahdishahr, IRAN. (djalicrt@gmail.com)
2- Department of Computer and Information Technology, Mahdishahr Branch, Islamic Azad University, Mahdishahr, IRAN.
2- Department of Electrical and Telecommunications Engineering Technology, Mahdishahr Branch, Islamic Azad University, Mahdishahr, IRAN.

I. INTRODUCTION

Peer-to-peer applications (P2P) are no longer limited to home users, and start being accepted in academic and corporate environments. While file sharing and instant messaging applications are the most traditional examples, they are no longer the only ones benefiting from the potential advantages of P2P networks. For example, network file storage, data transmission, distributed computing, and collaboration systems have also taken advantage of such networks.

The reasons why this model of computing is attractive unfold in three. First, P2P networks are scalable, i.e., deal well (efficiently) with both small groups and with large groups of participants. Second, you can depend more on the functioning of these networks, since they have no central point of failure and are more resistant to intentional attacks such as denial of service them. Third, P2P networks offer autonomy to its participants, allowing entering and leaving the network according to their interest and availability, and make their decisions without relying on external entities.

Although P2P networks can contribute to resource sharing and large-scale collaboration in geographically distributed environments with decentralized control and weak coupling, their diversification and dissemination are hampered by the current lack of security. According to [2], it is a challenge to make these networks secure.

By aiming for P2P networks to be widely adopted, they need to be protected against the action of malicious nodes. These can purposely provide incorrect responses to requests at both the application and the network level. In the

first case, returning non-truthful information in response to a search, in an attempt to censor access to certain objects. In the second, providing false information about routes, aiming to partition the network. In addition, attackers can perform other malicious activities such as traffic analysis (including on systems that seek to provide anonymity) and censoring on those who wish to provide high availability.

Another type of unwanted behavior, manifested by many users, is trying to gain more from the P2P network than it offers in return. This disparity can be expressed, for example, in the use of disk space, when the attacker wants to store data in the nodes of the network in a quantity much superior to that which he himself makes available to the system. A similar situation occurs when the malicious node refuses to use its restricted bandwidth to transmit an object, forcing the requester to retrieve it from some other replica.

Problems such as newly listed make security area one of the main fields of study in P2P networks. In this context, the aspects to be explored depend on the type of application and the degree of security required. The main aspects investigated (defined in a very general way) are the following:

- Availability: ensures that an entity is ready for use when needed;
- Authenticity: determines whether someone (or something) is, in fact, who (or what) claims to be;
- Confidentiality: protects data from unauthorized observation;
- Integrity: protects data against corruption, whether malicious or accidental;
- Authorization: restricts, based on rights, access to resources;
- Reputation: determines degree of trust in the other entities of a system;
- Anonymity: it keeps the identity of an entity unknown;
- Negotiability: overshadows data to protect the entity holding them from being held accountable;
- Non-repudiation: prevents an entity from denying responsibility for actions performed.

This paper covers P2P networks and security, very current issues that have received significant attention from both the scientific community and industry. In research, this is due to the number of challenges to be solved, while in industry, due to the great popularization of this type of application, the constant concern with information security and the socio-economic impact of its use. The paper presents an overview of search results and security-related technologies in P2P networks, supported by examples from the wide range of systems currently available and operating.

The rest of the paper is organized as follows. In Section 2, we review concepts of P2P networks, including their key characteristics, types of applications and architectures. In Section 3, a set of proposals to solve the main security problems identified in P2P networks is now described in more detail. Finally, Section 4, presents the final considerations and a sample of the research challenges to be overcome.

II. P2P FUNDAMENTALS

The recent literature presents excellent reviews on P2P, such as [1], [3] and [6], which define, categorize and exemplify P2P systems. In this section, we review only the main points of P2P or that have a reflection on their security, making the text self-contained and clear in relation to the adopted terminology, without being repetitive in relation to the recent literature in the area.

1. Key Features and Definition

There is no consensus in the literature about exactly what P2P systems are or what are the essential characteristics of such systems. Originally, P2P refers to a distributed architecture style that contrasts with the client / server: fully decentralized distributed systems, where all nodes are equivalent in terms of functionality and tasks they perform. This definition is purist and excludes several applications accepted today as P2P. More recently, P2P has been associated with a class of applications that take advantage of resources such as disk and CPU present on the edges of the Internet.

According to [4], the two key characteristics of P2P are:

- Direct sharing of resources between nodes, without the intermediation of a centralized server (although the use of servers is allowed in tasks with less computational or communication demand);
- Ability to self-organization, fault - tolerant, assuming variable connectivity and

transient population of nodes as standard, automatically adapting to failures in both network connections and computers.

Another key point in P2P is the notion that the system is built on the (supposedly voluntary) collaboration of the participants. Based on these characteristics, P2P systems can be defined as follows.

"Peer-to-Peer (P2P) networks are distributed systems consisting of interconnected nodes able to self-organize in topologies overlay in order to share resources such as content (music, videos, documents, etc.), CPU cycles, Storage and bandwidth, capable of adapting to transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a central entity."

In a P2P system, nodes establish logical connections and interact through them, forming an "overlay network" or overlay application level. In the rest of this text, the term "overlay" is used to denote a set of interconnected nodes in a P2P system, application, or network. In contrast, the term "network", unless otherwise stated, is used to denote the underlying physical network.

2. P2P Applications

Considering the key characteristics and definition of P2P presented, we have identified the following categories of P2P systems and / or applications:

• **File Sharing (sharing files):** One of the simplest but most popular P2P applications, its purpose is to allow users to "publish" files, whose content remains unchanged and is disseminated to any users, geographically spread around the world and potentially in large numbers. Typically, any user can publish a file on the system, and there are no read restrictions. Examples are Napster [10], Gnutella [5], KaZaa [8] and BitTorrent [9];

• **Networked File storage system (network storage):** unlike the previous case, the contents of the files can be modified by users (not immutable) and changes should, in the case of replication, are consistently propagated to all replicas. It is typically necessary to control and restrict write operations, and potentially read operations as well. Examples are PAST [12], OceanStore [12], Ivy [12] and JetFile [12];

• **Data Transmission or multicast overlay:** in this case, the overlay forms a communication infrastructure that supplies the absence of native multicast support in the network, in order to allow the same content to be transmitted by a node and delivered to a potentially large number of nodes (Users) geographically spread across the globe. Such technology has been used to broadcast live events. One of the main examples in this case is the ESM - End System Multicast [6], which has already been used to convey more than thirty events, including popular symposia such as SIGCOMM and INFOCOM;

• **Distributed Computing:** these applications are aimed at intensive processing performance through the idle capacity of exploitation (cyber-foraging) on computers that are part of grid systems. Examples are the OurGrid [3], Seti @ Home [13] and Genome @ Home [13]. Note that Seti @ Home and Genome @ Home have been categorized as P2P in the literature, but are based on a master / slave model where a central (master) server allocates work to personal computers (slaves). In these systems, access control and reputation management are two fundamental mechanisms and, thus, have received great attention from the research community;

• **Collaboration and communication between users:** these are applications that allow users to communicate through voice (VoIP), text messages, graphic images and general files in a direct way, without going through a server. Examples are Skype [7] and Instant Messaging applications such as ICQ, Jabber [7], MSN Messenger [7] and Yahoo Messenger [7].

In addition to the above categories, there are other applications that employ P2P concepts, such as distributed database manager system, but that are beyond the scope of this work. For purposes of terminology and treatment of the topic, we believe that a P2P overlay consists of a set of nodes (peers) connected through links dictated by protocols at the application level, in turn based on transport protocols. Nodes are identified solely on the overlay through an identifier (occasionally abbreviated id), can store and display objects (files, data blocks) to other requisites nodes, as well as offering services to other overlay nodes, including communication services Instant Messaging (IM) and processing for compute-intensive applications. In addition,

nodes collaborate in executing search protocols for objects and services originating from other nodes. Searches are based on routing tables and keys that uniquely identify an object or service in the overlay.

3. Overlay Organization

P2P systems can be classified into two main categories, as the overlay organization: **structured** and **unstructured**. The organization of an overlay has significant influence in several aspects, including security, robustness and performance.

Essentially, the organization determines the rules, if any, for allocating objects (or their keys) to nodes, and the search algorithms employed. In unstructured systems, the topology is determined ad hoc: the as nodes enter (and leave) the overlay, establish links with other arbitrary nodes. The positioning of objects or services is, in this case, completely independent of the overlay topology. One difficulty associated with this type of overlay is the process of searching for objects or services. Primitive methods such as flood overlay were employed in the first P2P systems, such as the original Gnutella. The inefficiency of this method has fostered research and development with an emphasis on scalability. As a result, there were no structured systems that employ more efficient strategies in terms of use of resources, such as random walk [2] or routing indices [5], and overlays scalable via Hash Tables distributed (DHT, distributed hash tables).

In structured overlays, the overlay topology is dictated by a key allocation scheme to node ids, in order to associate a given object or service to a specific node deterministically and globally known in the overlay. DHT functions as a distributed routing table, allowing an object to be found in a small number of steps. However, using DHTs requires a perfect match between the fetched key and the key offered as a parameter in the fetch; In other words, the requesting node must know the key of the searched object perfectly, and this is not always possible. In addition, some authors argue that maintaining overlay in highly transient populations is difficult. According to [14], structured overlays can be further separated into infrastructure and systems, depending on whether the overlay is only a scalable routing infrastructure at the application level (such as Chord, CAN, Pastry) or is it a Complete system (such as OceanStore, PAST and Kademlia). In the rest of this text, we do not distinguish between infrastructure and system, unless explicitly mentioned.

A structured overlay can be modeled through its key attributes. Combining the models of [12] and [17], we have the definition that follows. DHTs are typically consisting of a store API disposed on a search protocol layer. The latter has six properties: **P1:** a key area, with a key the unique identifier of an object, typically generated using a hash function such as MD5 or SHA1; **P2:** a space of node identifiers and a mapping scheme; for example, Chord uses circular space ids as CAN use dimension coordinate space d. A node ID could be the hash of its IP, for example; **P3:** rules for dividing identifier space between nodes: a DHT divides the complete space of node identifiers between existing (active) nodes at a given instant; **P4:** rules for associating keys with particular nodes, since each node is responsible for certain keys; **P5:** routing tables per node, and a routing scheme that populates the routing tables; **P6:** rules for updating tables in inputs and outputs of overlay nodes; When a node enters, it assumes responsibility for a portion of the id space belonging to other nodes (and consequently, the key space).

4. Top Unstructured Overlays

The main examples of P2P infrastructures that have an unstructured organization are described below.

Napster. Precursor in terms of file sharing, Napster was the "killer application" for dissemination of P2P culture. Despite this, Napster goes against the principles of P2P, because it depends on a central server for its operation. Napster has been hugely successful in allowing the sharing of music files. Predominantly, the content published on Napster was protected by copyright and may not be copied by other users; Today, the content provided by Napster is legal, but the same does not have as much appeal. Users who join the Napster network offer content by sending information about local files to the central server; Search operations are resolved on the server, which returns to the requesting node a list of node addresses providing the searched file. The download file then takes place directly between the nodes involved, without overloading the server. The Napster architecture is exemplified in Figure 1 (a): user computers search for information about files in the central

directory and then communicate with other nodes directly to obtain the desired files. According to studies in [6], Napster has more than 26 million users in 2001. Further information about Napster can be obtained in [2006].

Gnutella. It is a sharing system of ad hoc topology files. All nodes are functionally identical, said client because they are servers and clients simultaneously. File searches are performed through a flood of limited scope (called "horizon"). Nodes in which there is a marriage between the specified file name and the set of files published by the node send a positive response, by the reverse path in the overlay. The requisitor node then choose one of the nodes returned response and downloads directly from this node. There is no guarantee that a file will be located, but search performance is good for popular content. Figure 1 (b) shows an example of Gnutella architecture, where a node is a flood to find a file, and once found, download directly from one of the nodes that responded (positively). Improvements have been made in recent versions, through a hierarchy of two levels with super nodes responsible for index information from other nodes, as illustrated in Figure1 (c). In addition, the search scheme was modified in order to reduce the degree of flooding of the network. According to [16], in April 2016 there were approximately 2,219,539 users in the Gnutella network. Further information on the same can be found in [16].

FastTrack / KaZaa. It is a file - sharing system that employs a two - tier architecture, with normal and super nodes. Normal nodes connect to a super node, and super nodes connect to each other. A normal node maintains a list with addresses up to 200 super nodes, while a super node can maintain a list with thousands of super node addresses. When Is connected to the network, a node sends to its super node a list with the description of the files that it is making available.

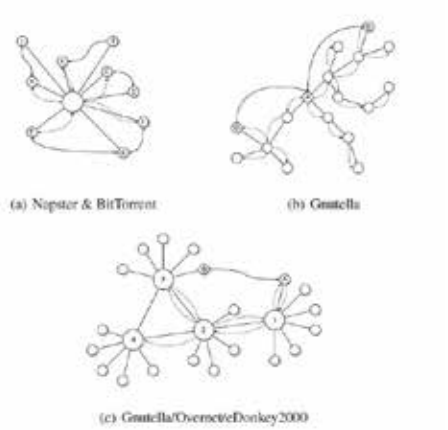

Figure 1 Examples of unstructured P2P[12]

A node sends a search to its super node, which either responds directly (when it knows the location of the desired file) or performs a search by sending messages to other super nodes. There is a certain degree of guarantee in the search of files, as the searches are sent to the super nodes, offering good performance for popular content [15]. In case of failure of a super node, the orphan nodes are passed to other super nodes. The FastTrack architecture is exemplified in Figure 1 (c) and resembles the current version of Gnutella: a normal node queries its super node for the location of a file, and if it exists, then requests the desired file directly to another node. According to statistics available in [6], in April 2006 there were approximately 3,144,691 concurrent users on the FastTrack network. Further information on FastTrack/KaZaa can be obtained from [12].

BitTorrent. File sharing system based on "clusters" (swarms) of nodes, exchanging files directly blocks but are coordinated by a central node, the tracker. The list of files and their properties (including a hash of each file), and the address of the tracker responsible for the swarm, are specified through a torrent file named. This file is prepared by a user who wishes to publish content, and made available on specialized websites; a torrent to be located before by interested users, which is looking on site with torrents or via web search engines.

Once uploaded the torrent file, the user provides the client software that connects to the tracker. The node informs the tracker about his interest in that torrent, and it answers with a random list of present nodes in the cluster. The node then contacts multiple nodes, requesting blocks of 512 KB from the file. BitTorrent uses

an incentive policy based on "eye-for-eye" (tit-for-tat), as described in [3]: a node has a number of neighbors in the cluster, and in principle all are "suffocated "(choked); then a node chooses a number (by default, 4) nodes among its cluster partners, that will upload. In BitTorrent terminology, it is said that these nodes will be unchocked. The choice is based on the download rate obtained from neighbors with whom the node interfaces, plus an optimistic unchoking where a node is chosen periodically and randomly. The tracker monitors the availability of nodes and pieces of the same object; Nodes are periodically tested, and a node that does not respond is successively changed. The architecture of a BitTorrent swarm is illustrated in Figure 1 (a): the central element is the tracker swarm, and it connects to several hundred users nodes that exchange data with each other. Note that there is only one central point, but an arbitrary number of trackers spread over the Internet and share responsibility for thousands of torrents. More information about BitTorrent can be found in [5].

Overnet/eDonkey2000. It is a hybrid architecture of two layers, composed of "client" nodes and "server" nodes, these responsible for indexing information about files and participating in search operations. Both client and server are run by any users. Figure 1 (c) presents an example of Overnet architecture. Recent versions of eDonkey implement the Kamdelia protocol [12], typical of structured overlays. According to [14], there were 3,736,358 concurrent users in this network in April 2006, but the eDonkey site [13] reported on the same occasion only 920,387 users. More information about Overnet / eDonkey2000 can be found in [13].

Freenet. It is a file sharing system with guaranteed anonymity. Weakly based on DHT, it uses keywords and descriptive text to identify objects. Searches are performed from node to node via keys or strings containing descriptive text. The routing ensures locate objects using a key until requests exceed the limits of hops-to-live. There is no hierarchy or central point of failure.

5. Top Structured Overlays

The main examples of P2P infrastructures that have a structured organization are described below.

Chord. Routing infrastructure that uses consistent hashing SHA-1 to associate object node keys in a circular space node ids m bits, i.e. 2 m identifiers. Identifiers nodes are obtained by making an IP address hash as key objects are obtained by making a hash of the description of the object. A key k is associated with the node identifier equal to k, or if it does not exist, the next node in the ring (said "successor node"). Consistent Hashing allows nodes enter and leave the network causing little stress to the P2P overlay; when a node joins the network after the node n, it assumes responsibility for a portion of the keys that were with n. Each node keeps a pointer to the node N immediately successors and a finger table with up to m pointers to other nodes (logarithmically around the ring). The routing search is unidirectional along the ring and may be recursive or iterative. In recursive mode, the message is forwarded from node to node and approaching the predecessor of the object; When it arrives at the node with the object, the search returns recursively to the origin. In the iterative, the requesting node is asking nodes that are getting closer to the node with the object; When the node with the object is asked, it responds with the data. Failure of nodes does not cause global failure, and replication of objects may occur in consecutive nodes. Figure 2 (a) illustrates an example of a Chord topology and a successful ring search operation. One of the examples of Chord use is the Cooperative File System, CFS [11]. Further information on Chord can be obtained from [12].

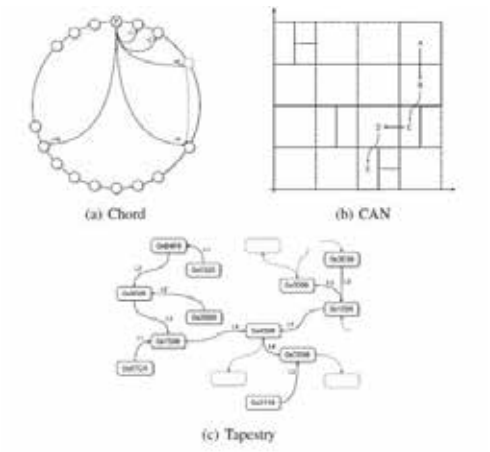

Figure 2Examples of Structured P2P[11]

CAN. Content-Addressable Network is a decentralized infrastructure whose basic principle is the use of a virtual Cartesian coordinate space of n dimensions.

In a multi-torus. The coordinate space is entirely logical and serves to implement node identification and its location via distributed routing tables. Each node is responsible for a space zone, which is dynamically determined, and maintains a routing table with the IP address and coordinates of each of its neighbors in space. The search protocol employs pairs <key, object> to map a point P in the coordinate space using a uniform hash function, and places these coordinates in the message. A message is routed to the destination using a simple route to the node that is closest to the coordinates. Node failure does not cause a global failure; Multiple nodes are responsible for an object, and when there is a failure, the application does a retentive. Figure 2 (b) illustrates the coordinate space of a CAN network and the routing of a message towards the destination node, which is responsible for the searched key. Further information on CAN can be obtained from [7].

Tapestry. Tapestry is a P2P infrastructure that allows message routing to objects (or the copy closest to them, if more than one copy exists) in a distributed, self-administered, fault-tolerant way. Routing and location information is distributed between network nodes. The consistency of the topology is checked dynamically and can be rebuilt in case of loss. Tapestry is based on mesh localization and routing mechanisms proposed in [15]. This distributed structure allows nodes to locate objects on an arbitrary size network using small, constant-length routing maps. In the original Plaxton mesh, nodes can assume the role of servers (which store objects), routers (which forward messages), and clients (which originate requests). Each node maintains a map of neighbors; each map has multiple levels, each level n containing pointers to nodes whose id must match in n digits. Each entry in the neighbor map corresponds to a pointer to the nearest node in the network whose id matches the number on the map, to a digit position. Messages are incrementally routed through the digit-by-digit nodes, from right to left. For example, a node message from 67493 to 34567 node might pass the following nodes: xxxx 7 -> xxx 67 -> 567 xx - x> 4567 -> 34567. The Plaxton mesh uses a root node for each object that serves as A guarantee from which an object can be located. When an object is inserted in the network node in us a root node nr is associated with the object using a global deterministic algorithm. A message is then routed nr ns for storing data in the form of a mapping <o, ns> in all nodes along the way. During a search operation, messages to the are initially routed to nr destination until a node is found containing the mapping <o, s>. Figure 2 (c) demonstrating a message routing example, extracted from [11]. More information about Tapestry can be found at [16].

Pastry. As the Tapestry is a routing infrastructure based on Plaxton mesh style. The main difference lies in the approach to obtain network location and replication objects. Pastry is used by persistent storage system large-scale PAST [1] and Scribe [3], a group communication system and communication events Large scale. More information about the Pastry can be found at [5].

Kademlia. Routing infrastructure that uses an innovative mechanism for routing messages and search for objects according to a metric distance between nodes identifiers (non-network proximity) based xor . The topology has the property that every message exchanged loads or reinforces contact useful information. The system exploits this information to send asynchronous and parallel search messages that tolerate node failures without imposing delays and timeouts to users. Several P2P applications are using the Kademlia algorithm: Overnet, eDonkey and eMule, and BitTorrent, which employs Kademlia to allow the use of torrents without a tracker. More information about Kademlia can be found at [7].

III. REPUTATION AND TRUST

The efficient and correct operation of a P2P system depends on the voluntary participation of its members. When the nodes of a P2P system do not cooperate, the consequences are serious and can cause damage to individual users or even cause the system to collapse. Traditional systems assume that users are “obedient” to adhere to a specified protocol without question the usefulness of it for you. This obedience is not a realistic assumption in P2P systems [6]. It

is therefore necessary that the control system in a way the nodes, blaming them for their actions when they refuse to cooperate, and rewarding them when they collaborate properly. Popularly, nodes that use more resources than offer (on offer) are known as free-riders or "ride nodes." Studies [10] show that free-riders are common in many P2P applications file sharing, and the explanation for this phenomenon is related to the "Tragedy of the Commons" [9], which argues that people tend to abuse the use of certain resources if they do not have to pay for them somehow.

According to [11] nodes of a P2P overlay that do not cooperate can be divided into selfish and malicious. The goal is get the most selfish P2P system contributing minimal resources for the same. In contrast, the purpose of malicious harm is one of the nodes or the system as a whole, as explored in the other subsections of this paper. Malicious nodes are willing to employ resources in the attack, such as injecting corrupted files, which does not occur with selfish nodes.

There are different approaches to encourage nodes to collaborate. [4] identifies two general classes of incentive mechanisms: based on trust (trust) and reputation, and based on trade (trade), including micro-payment mechanisms (in which a node overlay P2P offering one service to another is explicitly renumbered) and resource exchange schemes. A third category is mentioned in [15]: inherent generosity, where users decide to contribute to the system or not based on the overall level of contributions from other users.

A trade-based layout example is MojoNation [5], where users offering resources such as processing time and disk space can accumulate unit's mojo, and later, spend them. Another is the "network favors" [4] proposed to the grid system P2P OurGrid where the decision to accept or not a remote node task is taken under the previous history between the nodes involved.

Trust schemes and reputation are protected in reciprocity [11]: each node has an associated reputation, which starts from an initial state and is built over the life of a node, based on their interactions with other nodes. This reputation information can influence the decision of a node on the partners of their close interaction, seeking reciprocity. For example, a node may prefer to request a service to a node that has you running correctly and efficiently the latest requests or fail to consider certain nodes have returned files with corrupted content. Reciprocation is achieved in a P2P system through a system (management) reputation.

The terms reputation and trust (trust) are closely linked. There is considerable variation in the trust settings in the literature, there is no consensus on its meaning. According to [1], trust can be defined as a "firm belief in the competence of an entity to act reliably and securely on which to rely, within a specified context." According [1], trust is usually specified in terms of a relationship between an entity that trust, the guarantor, and one that is entrusted the depositary. Trust is the basis to allow a depositary use or manipulate features of a guarantor, or may affect the decision of a guarantor to use a service of a depositary. The degree of confidence in an operation is inversely proportional relation to its risk. On the relationship between the terms of a reputation management system determines the reputation of nodes based on the history of actions the same and allows opinions on the degree of confidence are formed around other nodes.

Trusts can be from one to another (trust a node to perform a service), from one to several (relying on a set of nodes with whom information can be securely exchanged), many - to - one (such as trust a leader), and several to several (when a group trust each other). One of the important properties of trust is transitive: if A Trusts B and B Trusts C, then it is possible that the trust (at least limitedly) in C. This confidence level can be expressed in discrete or continuous scale. In the first case, the figures would be as "low", "medium" and "high," while the second could be values in the continuous interval [0, 1] . But in the latter case as representing ignorance, considering that lack of knowledge and lack of confidence are very different things. One way expressed in the Model View of Josang [8], is to employ a triple values, c , d and i , which correspond to "belief", "disbelief" and "ignorance" with c + d + i = 1 (1 > = c , d , i > = 0).

One of the main challenges in P2P area is a reputable system design that can determine, accurately and efficiently, reliable Suitable values for nodes of a decentralized system of a large scale, and in that nodes enter and leave independently and at any time of the system, potentially under different identities, and can forge messages and identities. The confidence of a node to another is based, of course, the node's view of the reputation of another. Nodes acquire good reputation through successful interactions,

receiving a positive assessment by the involved nodes; for this reason, these systems are said based on feedback.

Many reputation management systems have been proposed seeking to solve this challenge. The literature is particularly rich in proposals; to illustrate, there are more than fifty articles on trust and reputation in P2P since 2001, many with similar or partially overlapping proposals. According to [16] reputation systems have in common three main parts: information collection, determination score and ranking, and response actions.

The collection of information is related to the identification scheme used, sources of information, information aggregation and the policy adopted for nodes that enter the no associated historical system (such nodes are said to strangers). In extreme caution, a node trusts as an information source only in their local information. A cautious user can increase the sources of information asking the opinion of others you trust based on a prior external relation, or a node can ask other nodes (nodes or neighbors who have interacted with success) on a particular node. If insufficient, a node A can request a node B (you trust), to ask the nodes where B relies on a C node, recursively and transitive way. Here there is a trade-off between security and performance: increase the number of asked nodes (seeking qualified information about other nodes) can adversely affect system performance.

The survey on trust in [16] discusses sources of information that a node can get on another node, as well as the strategies that a node can use. The following sources are listed: trust a priori with another node; external sources that are reliable; nodes that are reliable and that are a node away; nodes that are reliable and that are several nodes away; and a global reputation system.

The information obtained can be applied by a node with the following trust strategies: optimistically assume that all strangers are trustworthy until proven otherwise; ignore all strangers until they are proven reliable; investigate a strange asking trusted nodes; transitively propagate research through friends of friends; or use a centralized reputation system.

One of the fundamental problems of reputation systems is to ensure the validity of the information provided by other nodes. Therefore, it is natural that in determining the reputation score of a stranger, previous experiences of the node itself are valued in relation to the opinion of other nodes. A common approach in this regard is the use of weights: the reputation of an information given by a node is proportional to the reputation of this node. Information collected through transitive trust can be weighed according to the reputation of the less reputable node in the chain of trust; Alternatively, if the confidence values lie in [0, 1], then the value resulting from the multiplication of reputation scores each of the nodes.

About management of reputation scores, [3] presents two possible approaches to map actions of nodes in a non-negative score of reputation: one is based on credit and debit (for nodes that provide or consume content/service, respectively), and the other only on credit, but where credits expire naturally over time.

In terms of response share, a reputation management system provides service as a value of reputation on other nodes, which can then be used in different contexts in a P2P system. For example, when searching a node to perform reliably a particular service, it can use the reputation information to infer how likely the service is running properly. Another example is the search for an object (such as a file) and find multiple nodes as candidates for source, use the reputation value (plus the performance) in selecting the node or nodes to contact.

According to [5], there are two main issues that have been addressed in several studies in the literature:

As new nodes, said strangers should be treated reciprocity schemes (i.e. reputation schemes)?

Strategies that are based on indirect reciprocity are vulnerable to collusive behavior?

These issues are important in terms of reputation management system vulnerabilities. There are different ways to attack a reputation system; the three main attacks are discussed below.

The first attack is known as whitewashing and only occurs when nodes can exchange your identity easily (which is the case for many P2P systems). A node can leave the system and then return with a new identity in an attempt to get rid of any bad reputation that it has accumulated. If the nodes policy towards strangers is permissive, nodes can “use” the initial reputation, leaving the system and re-enter with new initial reputation. If a node cannot distinguish a correct new node of an old, then whitewashers can cause the collapse

of the system if no countermeasure is taken [4].

The second type of attack is to plot against reputation systems. This type of attack is often effective because in typical reputation systems a node should consult other nodes on the reputation of a third party. If many nodes are compromised, then nodes may provide false testimony, to increase the reputation of a malicious node, or to strike a correct node diminishing its reputation. In principle, the attacker should have massive resources, making much of the overlay nodes were his; however, in many P2P systems there is not a secure authentication scheme, enabling nodes acquire multiple false identities (Sybil creating nodes) with a single physical node, as explored in [5].

The third type of attack is the node traitor [6]. In such an attack, a node behaves appropriately for a while in order to build a good reputation, and then operates the system making use of it. This attack is especially effective when the nodes earn privileges as they gain reputation. An example of the traitorous attack is when a user on eBay [7] builds a reputation with many small value transactions, and then injures someone in a large value transaction. In systemic terms, a traitor node can arise not from a behavioral change of a user, but a change in the environment: for example, a perfectly correct client machine can be infected with a virus style Trojan horse, then I could randomly abuse good reputation of the node. Resistance to this type of attack can be increased by using the analysis of the recent history of a node [4].

Finally, as previously mentioned, the reputation of a node is typically used in the selection policy of a node to interact with. In [5] is characterized impact selection policies adopted by between responses P2P nodes that originate searches. Various policies are identified, among which include choosing the node advertising the best ability, that is, with the lowest delay, which is calculated for each node depending on the capacity upload and the current number of uploads simultaneous; random, where the customer selects a node at random; and file chunking: the client breaks the file into multiple blocks, and makes download simultaneously a piece of each node that announced the file.

In view of attacks, the worst of political choice of a node is one that selects the node that advertises itself as the best node. For example, if a performance information is easily falsifiable, a search will only be successful when no response is a malicious node as a malicious node will always be chosen. According to [15], reputation systems cannot solve this problem even when the errors of reputation system are minimal. Techniques based on randomness are effective to increase the resistance of a P2P system to attack. However, randomness negative impact on performance when attackers are not present.

IV. REPUTATION MANAGEMENT SYSTEMS

This subsection briefly describes representative examples of reputation management systems, building on the concepts presented in this paper.

1.XREP

In [17] authors proposes a scheme for reputation management for Gnutella. Unlike other approaches, the reputation of nodes is combined with the reputation of objects, increasing the resistance of the kind Sybil attacks. Reputations are cooperatively managed through a distributed polling algorithm to reflect the community's view of the risk of downloading and use of an object.

The protocol proposed by the authors extend the Gnutella search protocol with steps and additional messages, facilitating the assignment, sharing and combining reputations of nodes and resources.

The schema is the core XREP protocol, an extension to the Gnutella search protocol. Gnutella, a node initiates a search by sending messages (QUERY) to its neighbors; all nodes that meet the requested object return response messages via the same path by which they came. After receiving multiple hits (messages QUERY HIT), he asks other nodes opinions on these nodes that offered the sought object.

The reputations are binary, with (+) or (-), but the values can be both discrete and continuous. The protocol polling consists of five phases:

- Search features: at this stage, the messages QUERY HIT that are returned by the nodes that holds one or more objects that satisfy the search, adds a digest for each object referenced in the message;
- Selection feature and voting by poll: the requisitor node chooses the best node among those who seem to meet your search (i.e. respondents). To this end, the node sends a message to their peers

containing a polling request on the reputation of the offered objects and nodes that offer. These messages are implemented using conventional messages QUERY, and contains a public key to be used in response encryption to protect the integrity and confidentiality of responses. Nodes receiving the question check their repositories and respond;

- Assessment of the vote: the node discards garbled messages (the authors indicate that a node also makes a grouping and combination of votes that come from the same node to prevent Sybil attacks, but do not show how this could be done except with IP address) selects a set of voters and sends another message poll (TRUE VOTE) direct to each, to respond confirming their votes (this step requires nodes attackers to use real IPs);
- Check node better: the most trusted node is contacted to verify that it actually exports that object;
- Download object: node contact each other and requests the download of the object, after which it checks the integrity of the object through the digest and updates your experiences repository.

2. EigenTrust

EigenTrust [3] is an algorithm for reputation management for file sharing systems. Each node has associated with a global reputation, which is based on the file upload history. The global reputation of a node i is based on the reputation indexes locally assigned to i for each node j , k , l , etc. and weighted according to their own reputation of those nodes. In the study, the approach helped reduce the number of published spurious files.

The confidence values assigned by a node are normalized. This is necessary to prevent a node subvert the system by assigning arbitrarily high reputations to other malicious nodes, influencing global reputation for value in a collusion attack. In a node i.e., the reputation of the node j is normalized by dividing the reputation value j in i the sum of all reputation values that i holds. That is, all reputation values assigned by a node are between 0 and 1. The disadvantages of this are two standardizations. First, no distinction between ignorance and bad reputation. Second, the values are relative, and therefore cannot be interpreted in an absolute manner; for example, in i two nodes j and k have the same value reputation r, then it is known that the eyes of i , j and k are equally reputable but it is not known if both good and bad reputations.

To aggregate the normalized values computed for each node, a node i question its nodes "friends" j on the reputation of values that they assigned to a node k, and uses a weighted average to the reputations them to calculate the confidence i in k. To increase knowledge, a node can ask the opinion of friends of friends, and so on, recursively, until the entire network.

You cannot allow, of course, each node is responsible for calculating and reporting their own reputation. Therefore, the reputation of a node is computed by more than one node in the network, and stored in another node. Multiple nodes compute the score of a node, and a DHT is used to find such nodes. The proposed algorithm prevents a node to know the identity of the node to which it is calculating confidence, so that a malicious node cannot artificially increase the reputation of another malicious node. Nodes entering the system cannot choose which position you enter the ids space, preventing a re-enter the exact node in the node position responsible for calculating its reputation.

Global reputation values can then be used for isolation of malicious nodes. A node uses the reputation of the candidates and who will download a file selection policy. However, a choice of this kind concentrates requests to the nodes with higher reputation and does not allow other nodes to acquire correct reputation. The proposal is to use a scheme where the node is selected from semi-random, with influence of reputation.

According to [6] EigenTrust offers a purely decentralized solution, but uses a weak identity, making it susceptible to attacks whitewashing. Furthermore, [5] indicates that EigenTrust is susceptible to Sybil attack, since a malicious node can create an entire graph.

3. PeerTrust

PeerTrust [11] is a reputable framework that includes an adaptive trust model to quantify and compare the trust of nodes based on a system of transactions with feedback, and decentralized implementation of such a model in a P2P network. The two main features of PeerTrust are three basic parameters defining reliable and two adaptive factors in computing the confidence level of a node and the definition of general confidence metric to combine these parameters. Factors that

a node takes into account the confidence level calculation in PeerTrust are:

Feedback received from other nodes as a value; scope of feedback, such as the number of transactions that a node has with another;

Credibility factor for the node that provided the feedback differentiating quality feedback received from other nodes in accordance with the confidence in;

Transaction context factor to differentiate the most significant of the least significant associating weights to transactions such as taking into account the value of the transaction;

Community context factor to treat characteristics related to the community and particular vulnerabilities, such as creating an incentive for submitting feedback on other nodes.

For implementation of this trust model, each node has a trust manager and a data locator. The first is responsible for submitting feedback and trust assessment through a database with a segment of the global basis. But the data locator is for allocation and reliable location data in the overlay. The manager performs two main functions:

- Submit feedback to the overlay through the localizer, which routes the information to other nodes;
- Measures the confidence level of a particular node, which is performed in two steps: first, it collects reliable information about the node in question through the browser, and then computes the value of trust.

Two methods are proposed for confidence level calculation: Dynamic reliable calculation (DTM, dynamic trust computation), which uses "fresh" information obtained on demand from all other network nodes; and reliable approximate computation (ATC, approximate computation trust), which is more efficient but less accurate when calculating the confidence to present information in a cache. Each node maintains a cache containing the confidence values provided by other nodes that it recently announced; it only needs communication when not find a node in the cache.

The trust model uses recent transactions to calculate the confidence to avoid the traitor of the problem. When the reputation of a node is based on the cumulative average of their transactions of their lifetime, and a node has acquired a solid reputation, this time transaction has little impact on the reputation, and therefore a node has less incentive to behave honestly. A node can still oscillate between honest and dishonest behavior in order to maintain a reasonable reputation while acting incorrectly in certain transactions. The authors propose a simple al of sliding time window. The confidence values are computed globally and recently and compared. The idea is that a good reputation is hard to win, take time to build but can be destroyed quickly after a few incorrect transactions.

To avoid security problems related to the storage and transmission of reliable information, PeerTrust employs encryption with public / private keys. Each node is required to have a key pair and sign your messages feedback with your private key, and provide the public key, guaranteeing the integrity and authenticity. The id of each node is a digest of your public key, or your public key. To handle routing attacks [4] suggests the use of replication, but their detailed proposal.

5.4.3.4. TrustGuard

The authors in [5] discuss three attacks the reputation systems and how they can be counteracted with TrustGuard. The first is the traitor attack, in which a node accumulates good reputation and then change their behavior. Another attack is the shilling, where nodes provide feedback false and collude to increase their own reputation. The third is to flood the system with multiple feedback false about nonexistent transactions. In this sense, TrustGuard contributions are: introduce a trust model that deals effectively with strategic oscillations in the behavior of malicious nodes;

proposal for an admission control based on feedback to ensure that only transactions with secure evidence is recorded in terms of reputation;

Proposed credibility algorithms feedback to effectively filter feedback dishonest.

The architecture of the TrustGuard in each node, consists of three main components: A Reliability Assessment Machine (Trust Evaluation Engine), the Transaction Manager (Transaction Manager) and Reliability Information Store service (Trust Data Storage Service).

Before a node i establish a transaction with a node j , he asks the Confidence Rating machine to evaluate j. The machine uses an underlying DHT overlay to contact other nodes, collect feedback

and aggregate it into a reliable value.

The second component, Transaction Manager, takes as input values produced by the machine and makes trust decisions. Before executing a transaction, the Manager generates and exchange evidence of the transaction; Once the transaction is completed, the feedback is received by the two nodes involved. Messages with feedback are routed through the overlay DHT to designated nodes responsible for storing these values.

The designated nodes then invoke the third component, the storage service, which admit one feedback only if it passes through an admission control test for false transactions.

The problems and solutions addressed by TrustGuard are:

- Strategic or oscillation problem of traitors: the proposed solution is to incorporate in determining the confidence in a node fluctuation in its behavior, and the reputation history. If a node oscillates its reputation, then it negatively affects your reputation;

- Detection of counterfeit transactions: to prevent a node submit feedback transactions that never occurred, or positive themselves or other malicious node, or incorrectly negative about another node to be attacked, it is proposed that the nodes exchange transaction evidence the so that a node can demonstrate that really made a transaction with another. This prevents a node invent feedback on nodes that did not interact, but does not prevent a node submit feedback incorrect on a node who conducted a transaction.

- Feedback dishonest: to deal with the problem mentioned in the previous item, which is more serious in collusive situations, TrustGuard apply weights on the amounts reported in accordance with the reliability of the node that reports as EigenTrust and others.

4. FuzzyTrust

The authors in [5] paper describes a reputation management system for e-commerce P2P systems. FuzzyTrust uses fuzzy logic (fuzzy) to compute local scores confidence and make the aggregation of global scores. The system uses a DHT to exchange information on reputation (as in the case of TrustGuard).

The FuzzyTrust was designed based on an extensive analysis performed on transactions made on eBay [8]. The pattern follows a Power Law: there are few high-value transactions, though many transactions with small value. The study on the eBay gave rise to three design principles:

The bandwidth consumption can be quite high for exchanging reputation for hotspots (nodes that are involved in most transactions) and therefore should consider the transaction imbalance between nodes;

To deal with the least impact of certain users, the system should not apply the same evaluation cycle to all users, so that the most frequent users should be assessed more frequently; because some transactions are concentrated most of the value, it makes sense to assess the larger transactions more frequently than small ones.

FuzzyTrust performs reputation calculation locally and globally. Locally, nodes employ the inference engine fuzzy to capture uncertainties and if AutoFit the variation of local parameters. The aggregation of collected reputation scores of all nodes is made to generate an overall score for each node. Three aggregation weights are used as parameters: the reputation of a node, the date of the transaction, and the transaction amount. As a base, five rules fuzzy are used in the work described (the authors explain that a greater number could be used in a larger system):

If the transaction value is quite high and the transaction time is recent, then the weight in the aggregation is quite high;

If the transaction amount is quite low or the transaction time is quite old, so the weight in the aggregation is quite low;

The reputation of a node is good and the transaction value is high, then the weight in the aggregation is quite high;

The reputation of a node is good and the transaction value is low, then the weight of aggregation is average;

The reputation of a node is bad, then the weight in the aggregation is quite small.

Results are shown a comparison by simulation between the FuzzyTrust and EigenTrust using three metrics: convergence time required to establish the global reputation of each node; the detection rate of malicious nodes; and the overhead of messages involved in the global reputation of aggregation, presented individually for each node and globally. FuzzyTrust and EigenTrust have similar times for the global reputation of convergence. On average, with

FuzzyTrust fewer messages per node, and also global. FuzzyTrust has a good detection rate of malicious nodes, between 80% and 98%.

5. Trust Groups

In [4] proposes a framework for reputation management in large-scale P2P systems where it is assumed that all nodes are selfish, using a "virtual currency" for reputation measurement. The reputation of the nodes is decremented with the passage of time, so that nodes need to continue collaborating and providing services. The main contribution of this paper is the use of mutual trust groups that act jointly in relation to reputation. Nodes form communities that exhibit mutual trust and cooperate to combat selfishness and malicious behavior of other nodes.

Nodes with higher reputation should have easier access to services. When two nodes compete for a service, the provider node must choose the node that has a higher reputation. To get this, the system causes another node serving get higher reputation when the node is served higher reputation.

Nodes form trust groups, in which each member trust in others and uses this knowledge to defend themselves. The reputation of a node is the reputation of their group, which is determined by the average reputation of the nodes in the group. The reputation of a node only increases when it provides a service to a node outside the group. The division into groups provides increased security (the information is more reliable) and scalability.

the following attack models are treated:

a node always refuses to cooperate;

a node first cooperate and get reputation, and then passes uncooperative (traitor);

Malicious nodes make an attack in collusion aiming to reduce the reputation of certain correct nodes and cause the expulsion of the same group;
• nodes send fake certificates of satisfaction to increase the reputation each other, and offer bad service or bad when requested; with high reputation, can prevent the execution of certain tasks correct.

6. Freenet

Freenet [17] is a distributed storage system, which was designed in order to provide: (A) privacy for nodes that publish, retrieve and store objects; (B) resistance to censorship; (C) high availability and reliability; (D) storage and routing efficient, scalable and adaptive.

Freenet is implemented as an adaptive P2P network in which nodes perform requests each other to store and retrieve objects. These objects are identified by location independent keys. Each node maintains its own storage space and makes it available to the network for reading and writing. The nodes also maintain a dynamic routing table containing addresses of other nodes and key objects they store.

The keys in Freenet are calculated using operations hash SHA-1. Two types of keys are accepted:

Content-hash key: generated from the hash object to be stored, it is useful to verify the integrity of the object;

Signed-subspace key: generated from operations hash and XOR (a) a descriptive text of the object to be stored and (b) the public key associated with the namespace defined by the user you want to insert the object in the system.

Recovery and inserting objects. When a node receives a request, it checks locally and is the object, returns with a tag identifying himself as the holder of it. Otherwise, the node forwards the request to the node, listed on your table, which stores the closest to what you requested key. This process repeats until the request reaches the node that owns the object. At that time, the object is passed along the same path by which transited request, making each intermediate node to update its routing table associating the object holder with the respective key. To provide anonymity node that is storing the object, each node along the route may decide to amend the return message stating that he or any other is the source of the object.

Have to insert an object in the system, the requesting node attaches to a key object and sends a message INSERT for himself. This message includes a key and a value of hops-to-live that indicates the number of copies of the object store. The process to set the location where the object is stored is similar to a search; the message INSERT runs the same way as a request for the same key would travel.

If the amount of hops-to-live reaches 0 and no collision is detected, a message ALL CLEAR is sent to the node that requested the insertion. This node then sends the object that is being stored by nodes along the same route where the message INSERT passed beforehand. To provide anonymity to the node that is publishing the

object, each node along the route may decide to change the insertion message stating that he or any other is the source of the object.

On the other hand, if the key is already being used by another object, the node returns the pre-existing object as the node that requested the insertion had made a request for it. Thus, malicious attempts to replace existing legitimate objects for garbage will result in further spread of legitimate files already stored.

To reduce the amount of information that a malicious node can get by accessing the value of hops-to-live, messages do not fail to follow through when hops-to-live reaches 1 being forwarded with a certain probability (with hops-to-live worth always 1).

Anonymous communications. The privacy in the system is obtained by using a scheme similar to networks mix Chaum for anonymous communications [1]. Instead messages are transported directly from source to destination, they go through node to node chains in which each channel is encrypted individually, until the message reaches the recipient. As each node in the chain knows only its immediate neighbors, there is no way to determine the identity of the nodes publishing, the nodes that are storing objects and nodes from where the requests to retrieve objects.

Denial of stored objects. For legal and / or policies in order to provide the appearance deniability to the nodes, all stored objects are encrypted. The encryption procedures used do not aim to make the confidential object content, since any requesting node should be able to decrypt it to get it back. Rather, the goal is that the operator node can deny knowledge of the object's content, since he knows only the key of the object and not the key used to encrypt it.

Cryptographic keys objects stored with keys signed-subspace can only be obtained by reversing the operation hash. Already the cryptographic keys for objects stored with key content-hash are completely unrelated. So, just a brute force attack would allow the node operator has access to the content of the objects it stored.

7. Free Haven

Free Haven [3] is a distributed storage system based on a community called nodes client. In this community, each node hosts objects from other nodes in exchange for an opportunity to store your own objects that network later. The network consists of the client is dynamic: objects migrate from one node to another frequently, considering the trust that each node has with the others. The nodes transfer objects from negotiations between them (trading).

Each node has a public key and one or more return blocks [8], which together provide secure communication, certified and pseudonymous with it. Each node in client has a database containing the public key and return blocks of other servers.

Objects are divided into shares and stored in different nodes. Nodes that publish assign an expiration date to published objects. The nodes are committed to keep the shares of a particular object until its validity expires. To encourage honest behavior, some nodes check whether other nodes discard their shares before the combined and decrement their trust in these nodes. This confidence is monitored and updated by a system reputation. Each node maintains a database of trusted values (reputation) of the remaining nodes.

Recovery and inserting objects. To insert an object into the system, the node uses to publish the algorithm for dispersing information proposed in [9] for dividing the object in shares f1 , ..., fn where any k shares sufficient to recreate object. Then, the node generates a cryptographic key pair, selects and signs a segment object to compose each share fi, and enters those shares resulting in your local storage space. The attributes stored next to each share are: timestamp, expiration date, public key used to sign it (for integrity check) number share and signature.

Each object in Free Haven system is indexed by the key M corresponding to the hashed public key used to sign the shares that make up the object. To perform the search for an object, the requesting node generates a cryptographic key and a return block. Then sends a route request message broadcast informing the key H, the public key PK client and the return block. The message is received by all nodes that the requesting node knows.

If so, the node number each share using the public key PK client and sends it through the remailer to return block reported in the request message. These shares will reach the requesting node; at the time k or more shares have been received, the node can re-create the object.

Anonymous communications. Communication between nodes occurs via a network for sending and receiving e-mails anonymously (remailer network) [5]. Each node of the Free Haven system

has associated with it one or more return blocks (reply blocks) that network. These blocks consist of routing statements that identify how to get to a recipient. These instructions are encrypted successively to a set of remailers, so that each Remailer can identify only the identity of the next hop. The instructions in the core block, visible only by the last remailer reveal the final destination of the message.

Trading shares. Nodes exchange shares with each other periodically. The reasons for this negotiation unfold in four. First, to provide greater anonymity to the node that publishes; if changes occur with great frequency in the system, there is no way assume that a node is proposing an exchange node that publishes share what he has to offer. According to enable the input and output nodes; the idea is that a node that wants to leave the system negotiates with other nodes targeting locally store only shares short so that after expiry, can "get out" smoothly. Third, to allow storage of long objects with expiry dates. Fourth, to prevent static targets - for example a node storing certain shares - can be attacked aiming cause a denial of service.

Trading ends with the exchange of confirmation messages, accompanied by "receipt", between the nodes involved. In addition, each node sends receipts (a) the buddy linked to the share which is crumbling and (b) the buddy of the share that will store. The concept of buddy is used to provide reputation Free Haven and will be explained below.

Reputation. Malicious nodes can accept to receive certain shares and purposely fail when storing them. To avoid this kind of behavior, the system proposes the association between pairs of shares of the same object. Each share is responsible for maintaining information about the location of the other share or buddy. Periodically, the node responsible for a share sends a request to the node that is storing their buddy to make sure that it continues to exist. If it does not respond, the node that made the request announces the problem occurred, revealing the identification - in case a pseudonym - the node that was responsible for the storage buddy.

Given this and other opportunities that a node has to take advantages over the other, the Free Haven system uses a reputation mechanism which is to identify and account for misbehaving nodes. Each node maintains two information about the other: reputation and credibility. The first refers to the degree of confidence that a node is complying with the specification of Free Haven protocol. The second is the belief that the information received from that node are true.

The nodes spread "references" to the other whenever register the successful completion of negotiations, suspect that the buddy of a particular share was lost or when the reputation and credibility of values for a given node change substantially.

Attacks on anonymity. A set of attacks can be carried out in order to reveal information about the identity elements of the system, compromising their anonymity:

Attacks on anonymity reader: an attacker can develop and publish the Free Haven system a kind of virus that automatically contacts a particular host to be performed, revealing information about the node that the recovered object. Another attack is to become a node in both client and in mixnet and attempt an end-to-end attack, correlating, for example, the traffic of messages with the request by objects. Still, a compromised node may disclose which has given object and see who requests the same or simply monitor request messages by objects and store their origin. From there, it would be possible to determine system usage profiles and frame users to them. According in [1] Free Haven prevents this type of attack by employing each return block for only transaction.

Attacks on the anonymity of the node that stores objects: an attacker can create shares large and purposely try to reduce the set of nodes known for their ability to store these shares. Such an attack partially compromises the anonymity of these nodes. An attacker can also assume the role of one of the nodes, and if so, to collect information on the status and participation of other nodes in the system (e.g. lists of nodes). Finally, a simple but very damaging attack is the spread of a worm in the system, which identifies the objects stored in the nodes and informs them to an external application.

Attacks on the anonymity of the node that publishes: the attacker can assume the role of a node and register publication actions, seeking to associate source/origin and time. Alternatively, the malicious node can observe nodes that potentially recently published objects and try to determine who was communicating with them in the same period.

Attacks on reputation aimed at compromising the mechanisms associated with the identification

and accounting of malicious nodes. In the case of Free Haven system, the main attacks on reputation are as follows.

Simple treason: the attacker can become part of the network clients, behave properly long enough to build a good reputation and then move to act maliciously deleting locally stored objects before its expiration date.

Comrade pickup: if a malicious node or collusion obtains control over the share and its respective buddy, it can delete both without this action reflected in the system.

False recommendation (false referrals): a malicious node can disseminate "references" false network or send them to a subset of nodes, compromising the reputation of calculation performed on them.

Trapping: a malicious node may violate the Free Haven protocol in several ways. When another node detects this bad behavior and accuses him, the malicious node can present receipts that contradict the informer, and report the correct node to send "references" false.

V. CONCLUSION

In this paper we present a summary of the main safety aspects to be considered in P2P networks, highlighting its importance for the development of P2P applications and systems on the Internet and deployment of enterprise applications with more critical needs in terms of security.

P2P systems are no longer limited to home users, and start being accepted in academic and corporate environments. For example, network file storage systems, data transmission, distributed computing and collaboration have also taken advantage of these networks. This computing model is attractive because a number of reasons. First, because P2P networks are scalable, because they have no central point of failure or neck, in the form of a central server. Second, because they resist better the intentional attacks such as denial of service them. Third, because it has the power to attract a large number of users from the benefits offered by the community but without giving up the autonomy of its participants.

One of the main challenges in P2P is to provide guarantees for safe operation of P2P applications in decentralized settings and large-scale, crossing multiple institutional domains and users congregate and corporations with goals and demands so distinct.

The aim that P2P networks are widely adopted, they need to be protected against the action of malicious nodes. We present various types of vulnerabilities, attacks that exploit, and proposals of defense mechanisms to render such innocuous attacks. Examples of vulnerabilities discussed in this paper are attacks on the routing system (and the possible repercussions for the overlay), communications anonymity attacks and attacks on reputation systems. Problems such as these make the security area one of the main fields of study in P2P networks.

REFERENCES

[1] Schollmeier, R. (2011, August). A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In Peer-to-Peer Computing, 2011. Proceedings. First International Conference on (pp. 101-102). IEEE.

[2] Cai, M., & Frank, M. (2014, May). RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In Proceedings of the 13th international conference on World Wide Web (pp. 650-657). ACM.

[3] Ripeanu, M. (2011, August). Peer-to-peer architecture case study: Gnutella network. In Peer-to-Peer Computing, 2011. Proceedings. First International Conference on (pp. 99-100). IEEE.

[4] Ripeanu, M., Foster, I., & Iamnitchi, A. (2012). Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. arXiv preprint cs/0209028.

[5] Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., & Lim, S. (2015). A survey and comparison of peer-to-peer overlay network schemes. IEEE Communications Surveys & Tutorials, 7(2), 72-93.

[6] Bawa, M., Garcia-Molina, H., Gionis, A., & Motwani, R. (2013). Estimating aggregates on a peer-to-peer network. submitted for publication.

[7] Rowstron, A., & Druschel, P. (2011, November). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (pp. 329-350). Springer Berlin Heidelberg.

[8] Saroiu, S., Gummadi, P. K., & Gribble, S. D. (2011, December). Measurement study of peer-to-peer file sharing systems. In Electronic Imaging 2012 (pp. 156-170). International Society for Optics and Photonics.

[9] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., & Balakrishnan, H. (2013). Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transactions on Networking (TON), 11(1), 17-32.

[10] Freedman, M. J., & Morris, R. (2012, November). Tarzan: A peer-to-peer anonymizing network layer. In Proceedings of the 9th ACM conference on Computer and communications security (pp. 193-206). ACM.

[11] Kuhn, M., Szklarczyk, D., Franceschini, A., Campillos, M., von Mering, C., Jensen, L. J., ... &