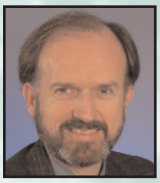
ISR Institute for Software Research
University of California, Irvine

A Survey of Trust Management and Resource Discovery Technologies in Peer-to-Peer Applications

Girish Suryanarayana
University of California, Irvine
sgirish@ics.uci.edu



Richard N. Taylor
University of California, Irvine
taylor@uci.edu

July 2004

ISR Technical Report # UCI-ISR-04-6

Institute for Software Research
ICS2 210
University of California, Irvine
Irvine, CA 92697-3425
www.isr.uci.edu

A Survey of Trust Management and Resource Discovery Technologies in Peer-to-Peer Applications

Girish Suryanarayana, Richard N. Taylor
Institute for Software Research
University of California, Irvine
{sgirish,taylor}@ics.uci.edu

ISR Technical Report # UCI-ISR-04-6

July 2004

Abstract: Decentralized peer-to-peer (P2P) applications are characterized by the absence of a central authority or infrastructure that coordinates the behavior of entities in the system. These entities, called peers, interact directly with each other and make local autonomous decisions in order to achieve their individual goals. In the absence of a single authority that maintains all the data and handles all the queries, peers themselves are responsible for seeking, storing, and sharing information efficiently. Placing a large amount of information on every peer or broadcasting a request to every peer greatly reduces the performance and efficiency of the system. Hence, it is essential that decentralized applications employ efficient storage mechanisms and reliable search mechanisms. Further, an open decentralized system that does not regulate the joining of peers can be subject to grave risks. In particular, malicious peers may be encouraged to resort to a variety of attacks, including sending spurious information, posing as other peers, etc. It is important for each peer in the system to defend against such attacks. This survey discusses these two essential issues that characterize P2P decentralized applications: storage and discovery mechanisms, and trust management. It identifies and defines key properties for each of these and also summarizes the efforts of the P2P community in addressing these properties by categorizing and discussing relevant technologies and approaches.

A Survey of Trust Management and Resource Discovery Technologies in Peer-to-Peer Applications

Girish Suryanarayana, Richard N. Taylor
Institute for Software Research
University of California, Irvine
{sgirish,taylor}@ics.uci.edu

ISR Technical Report # UCI-ISR-04-6

July 2004

1 Introduction

The recent success of file-sharing applications has resulted in significant attention to peer-to-peer (P2P) technology. The primary goal of these file-sharing applications such as Napster [Napster] and Gnutella [Gnutella] is to aggregate resources and present them to a user [Milojicic, Kalogeraki et al., 2002]. This enables users at the edge of a network to share and access files and is the main reason behind the success of these applications. Although the initial focus was thus centered mainly around file-sharing applications, researchers have since then realized the tremendous potential of P2P technologies and approaches. As a result, these technologies have been effectively applied to various distributed applications. While the first wave of research mainly concentrated on realizing the benefits of the P2P approach in different domains and applications, the recent second wave is more focused on enhancing the qualities of these P2P applications.

P2P applications can be classified into the following two categories depending upon whether they require the presence of a controlling authority [Cohen and Shenker, 2002]: decentralized P2P systems and centralized P2P systems. Decentralized P2P applications consist of a group of entities, called peers, that interact with each other without the presence of a central coordinating authority [Suryanarayana and Taylor, 2002]. A peer can act both as a client and a server, since it can request services from other entities as well as provide services to other entities in the system. Each peer has a limited perspective of the system and relies upon information received from other peers in the system to make local autonomous decisions. Decisions made by each decentralized peer may well conflict with those made by other peers. We term such non-consensus-based applications decentralized [Khare, 2003]. Reliable communication, and completeness and accuracy of information are imperative to the success of such P2P decentralized applications. There are several advantages of such a decentralized architecture including increased fault-tolerance and robustness, and enhanced scalability.

Centralized P2P applications consist of one or more special peers that coordinate and control the behavior of other peers in the system. Napster is an example of such a centralized P2P application because though it facilitates the direct exchange of music files between peers, it

maintains a centralized index of the content of each peer. The main shortcoming of centralized P2P applications is that these special peers can act as possible single-point-of-failures [Foner, 1997] that can reduce the reliability and performance of the system. Since our focus in this survey is primarily on decentralized P2P applications and solutions, an indepth discussion of centralized P2P applications is out of scope. However, our survey does examine a few of these technologies and compares them against decentralized technologies.

In the absence of a single authority that maintains all the data and handles all the queries, it is imperative that decentralized peers be able to seek, store, and share information efficiently. Placing a large amount of information on every peer or broadcasting a request to every peer greatly reduces the performance and efficiency of the system [Chawathe, Ratnasamy et al., 2003]. Hence, it is essential that decentralized applications employ efficient storage mechanisms and reliable search mechanisms. Further, an open decentralized system that does not regulate the joining of peers can be subject to grave risks. In particular, malicious peers may be encouraged to resort to a variety of attacks, including sending spurious information, posing as other peers, etc. It is important for each peer in the system to defend against such attacks.

This paper surveys these two essential issues that characterize P2P decentralized applications: storage and discovery mechanisms, and trust management. It identifies and defines key properties for each of these and also summarizes the efforts of the P2P community in addressing these properties by categorizing and discussing some relevant technologies and approaches.

The structure of this paper is as follows. The next section describes the two specific areas of decentralized P2P applications that we have identified. Section 3 and 4 discuss relevant technologies and approaches belonging to these two topics and compare them using topic-specific properties. This is followed by conclusions in section 5.

2 Essential Aspects of Decentralized P2P Applications

Decentralized peer-to-peer applications are composed of a set of entities, called peers, that directly interact with each other. These applications are characterized by the absence of a single centralized authority that is responsible for controlling and coordinating the behavior of the peers. Instead each peer relies upon data received from other peers to make local autonomous decisions regarding its behavior. These decisions may differ and even conflict with those made by other peers in the system. Further, depending upon the type of application, data exchanged between peers may vary in its type (e.g., resource files, information) and its value to the peers.

An open decentralized system that allows any peer to join the system at any time poses significant risks to the peers. In particular, malicious peers may resort to a variety of attacks, including sending spurious information, posing as other peers, etc. [Suryanarayana, Erenkrantz et al., 2003]. Since there is no central authority that can authenticate and guard against the actions of such malicious peers, it is up to the peer to protect itself from the effects of these actions. Consequently, each peer in the system needs to somehow evaluate information received from another peer in order to determine the trustworthiness of both the information as well as the sender. This can be achieved in several ways such relying on direct experiences or acquiring reputation information from other peers [Resnick, Zeckhauser et al., 2000].

Since peers depend on information and resources received from each other, an efficient and reliable resource discovery mechanism is required. However, the decentralized nature of the system poses certain challenges to such a discovery scheme. In particular, peers may leave and enter the system at any point in time potentially affecting resource availability as well as leading to stale or un-indexed resources. This decreases the effectiveness of the search mechanism since search results may either return stale data or fail to locate new data. Additionally, peers may be limited by their storage capabilities. This is significant because unlike a centralized system where all data can be stored on a central server, each peer may have to store a considerable portion of the global information locally. Therefore, it is important that data is stored efficiently across the peers. Peers may also be limited by their connection capacities thus restricting the amount of traffic they can handle. In such a case, broadcasting query messages indiscriminately will slow down the system. While this may satisfy the needs of a small system, it does not scale as the system increases in size [Ritter]. Therefore, it is essential for peer-to-peer decentralized applications to employ efficient and reliable search mechanisms.

Table 1 compares trust models with respect to the threats of decentralization while Table 2 compares trust models against their other properties. A comparison of discovery mechanisms against their properties is summarized in Table 3. For all the tables we use the following representation format. Columns represent models or mechanisms, and rows represent the properties they are being evaluated against. A value or symbol in a cell indicates either the presence or absence or the degree to which a particular model exhibits the corresponding specific property. If a property is not applicable to a certain model, the corresponding cell value is “NA”. A cell value of either “Yes” or “No” represents the presence or absence of the specific property in the model. Values for properties such as scalability and reliability are expressed using ‘*’ symbols. The number of ‘*’ in a cell denotes the relative extent to which the particular model exhibits the specific ability. It should be noted that the main purpose of

the tables is to strictly serve as visualization tools, and these tables do not contain any description of the various models and properties.

3 Trust & Reputation

The concept of trust is not new to us nor is it limited only to electronic entities. In fact trust is an integral part of our social existence. Our interactions in society are influenced by the perceived trust worthiness of other entities. It thus plays an equally important role in our daily lives. Naturally, in addition to computer scientists, researchers from other fields such as sociology, history, economics, and philosophy too have devoted significant attention to the issue of trust [Marsh, 1994]. Given the fact that trust is a multi-disciplinary concept, there exist in the research literature several definitions of trust and discussions about the factors that determine trust. While an in-depth discussion of these topics is outside the scope of this report, below we summarize some common definitions with an aim to provide a sufficient background for the purpose of this survey.

One of the most popular definition of trust is the one coined by Deutsch [Deutsch, 1962] which states that:

(a) an individual is confronted with an ambiguous path, a path that can lead to an event perceived to be beneficial or to an event perceived to be harmful; (b) he perceives that the occurrence of these events is contingent on the behavior of another person; and (c) he perceives the strength of a harmful event to be greater than the strength of a beneficial event. If he chooses to take an ambiguous path with such properties, he makes a trusting choice; else he makes a distrustful choice.

An interesting fact about the above definition pointed out by [Marsh, 1994] is that trust is considered to be subjective and dependent on the views of the individual. Deutsch further refines his definition of trust as *confidence that an individual will find what is desired from another, rather than what is feared* [Deutsch, 1973]. This definition is also echoed by the Webster dictionary which defines trust as *a confident dependence on the character, ability, strength, or truth of someone or something*.

Luhmann [Luhmann, 1979] approaches trust sociologically and considers trust as *a means for reducing the complexity of society; complexity created by interacting individuals with different perceptions and goals*. This definition because of its social nature is more apt for reputation-based systems.

Another popular definition of trust that has also been adopted by computer scientists is the one coined by Diego Gambetta [Gambetta, 1990]. He defines trust as *a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action*. Gambetta introduced the concept of using values for trust and also defended the existence of competition among cooperating agents.

A recent definition of trust has been put forth by Grandison and Sloman [Grandison and Sloman, 2000] who define trust as *the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context*.

While there are several aspects to trust, we outline below a few that we believe facilitate a better understanding of the concept of trust. Trust is conditionally transitive. This means that if

A trusts B and B trusts C, A trusts C only if certain possibly application-specific conditions are met. Trust can be multi-dimensional and depends upon the context. For example, A may trust B completely when it comes to repairing electronic devices but may not trust B when it comes to repairing cars. Trust can also be expressed in different ways such as a set of continuous values between 0 and 1, or binary values, or a set of discrete values.

Related to trust is the concept of reputation. Abdul-Rehman [Abdul-Rahman and Hailes, 2000] defines reputation as an expectation about an individual's behavior based on information about or observations of its past behavior. In online communities, where an individual may have very less information to determine the trustworthiness of others, their reputation information is typically used to determine the extent to which they can be trusted. An individual who is more reputed is generally considered to be more trust worthy. Reputation can be determined in several ways. For example, a person may either rely on his direct experiences, or rely on the experiences of other people, or a combination of both to determine the reputation of another person.

3.1 Trust and Reputation Properties

3.1.1 Local Control

Decentralized applications are typically composed of autonomous peers that typically have complete local control over their data. This data includes trust and reputation values along with other information and resources. Depending upon the trust model being used, this trust data may be a part of the global trust data. In such a case, the trust data maintained by a peer is not limited necessarily to its own perception of the trustworthiness of other peers. This may in fact offer a malicious peer the opportunity to change the data according to its best interests. The only way to prevent this in such trust models is by restricting a peer's autonomy, for example, by disallowing it to change the trust data it maintains. We use the Local Control property to distinguish such trust mechanisms that do not allow peers to change local trust values from those that do.

3.1.2 Trust and Reputation Values

Trust models typically use trust and reputation values to represent the trust one peer has in another. These values may either be discrete or continuous values depending upon the needs of the application and the type of trust model used. Some trust models also employ binary values, implying that a peer either completely trusts or distrusts another peer. For example, XREP (section 3.2.2.2) uses binary levels of trust while NICE (section 3.2.2.4) uses continuous values. Tagging peers as either completely trustworthy or untrustworthy does not permit a peer to express partial trust in other peers. Continuous trust values on the other hand provides a peer greater expressive power to define its trust relationships with other peers. We use this property to distinguish trust models on the basis of the type of trust and reputation values they employ.

3.1.3 Type of Reputation

This property indicates the type of reputation mechanism used by a trust model. In reputation-based trust models, peers may use three kinds of reputation information to determine the extent of trust in other peers: positive reputation, negative reputation, or a combination of both. In a positive reputation mechanism, peers only exchange information

about successful transactions. In a negative reputation mechanism, on the other hand, peers are generally assumed to be good and reputation is expressed only as negative feedback or complaints that are distributed to other peers. Both mechanisms while useful are incomplete by themselves. For example, relying only upon successful transactions may result in peers ignoring the recent malicious actions of a good peer. The drawback of relying only on a negative reputation-based scheme is that a peer may end up trusting a malicious peer if it does not have access to existing complaints [Damiani, di Vimercati et al., 2002]. We believe that a combination of positive and negative reputations makes the trust mechanism more robust and reliable.

3.1.4 Signature Verification

This property is used to distinguish trust models that explicitly use credential verification to establish the authenticity of the message originator. Credential evaluation prevents malicious peers from taking on the identity of other peers (impersonation). In one common technique every peer generates a public and private key pair. Any message sent out is signed by the sender's private key and authenticated by the receiver using the sender's public key. While credential verification can be easily added onto most trust models, a lot of trust models do not explicitly specify whether their trust models actually employ signature verification to establish peer authenticity.

3.1.5 Anonymity

Anonymity is an important consideration for open decentralized peer-to-peer applications [Freedman and Morris, 2002]. Protecting the identities of peers shields them against certain malicious actions. For example, if a malicious peer recognizes the real user behind a peer, it may try to spread spurious information about that peer. Guarding the privacy is therefore imperative and grants peers an additional level of protection against malicious peers. It is interesting to note that decentralization also facilitates anonymity. For instance, using intermediate peers as proxies during routing prevents the responding peer from knowing the identity of the initiating peer [Reed, Syverson et al., 1996].

There exist a number of protocols and infrastructures that aim to preserve the anonymity of peers [Shields and Levine, 2000; Scarlata, Levine et al., 2001]. However, they all ignore trust relationships between peers. This seems essential since there is an inherent trade-off between trust and anonymity. In order to be able to trust another peer, a peer needs to know the identity of the other peer. It is difficult to establish a trust relationship with an anonymous peer. This is why none of the trust models discussed in the survey achieve complete anonymity (see Table 1).

3.1.6 Bandwidth Cost

Peers in a decentralized application communicate by exchanging messages. Depending upon the nature of the application, it is possible that a lot of message traffic is generated. If, in addition, peers need to exchange information regarding trust, it results in higher bandwidth usage. This is exacerbated if this trust information is large in size or uses a number of messages. In particular, expecting peers to pass around long trust histories to other peers results in a lot of unnecessary bandwidth overhead and adversely affects the scalability and performance of the system. Therefore, reduction in bandwidth is an important objective of any trust model.

3.1.7 Storage Cost

It is possible that a trust model may require peers to store trust data about other peers. This trust data may have different forms – from trust values to detailed prior transaction information. Depending upon the type of information stored, it is possible that the trust model may require a peer to invest a significant amount of storage. We denote this cost incurred as the storage cost. In trust mechanisms where the trust data stored is proportional to the size of the system, this cost increases linearly as the number of peers in the system increases.

3.1.8 Fault-Tolerance

One of the main characteristics of a decentralized system is its constantly changing topology. This is typically due to the transient nature of peers which may enter, leave or be disconnected from the system at any time. Fault tolerance in this context represents the ability of the trust model to adapt to this transient nature of the system. When peers enter or leave the system, not only do new trust relationships need to be formed but trust values and transaction information may also need to be replicated across peers to ensure availability of trust data. This is essential because non-availability of trust information may lead to peers trusting malicious peers. However, a disadvantage of enforcing trust information replication is that a peer may lose privacy over its personal trust data, causing the system to not be truly decentralized.

3.1.9 Scalability

Scalability refers to the ability of the trust model to scale with an increase in the number of peers. An increase in the number of peers results in the formation of more trust relationships among peers. This leads to an extra storage and computation overhead at each peer that must now maintain a greater number of trust relationships. This overhead, in addition to maintaining more indices and routing information, needs to be addressed by the trust model. A natural effect of having a greater number of peers is also a potential increase in the number of queries for trust information. This builds up network traffic and depending upon the trust model may also increase the computation load on each peer. Thus, all these factors together determine scalability.

3.1.10 Reliability

This property embodies the ability of a trust model to help peers to correctly determine their extent of trust in other peers based upon their past experiences and/or information received from other peers. The trust model should help peers identify and successfully defend against spurious information, including wrong trust values, propagated by malicious peers and take corrective actions against them. Corrective action may involve educating other peers about malicious peers and nullifying the effects of spurious information. Additionally, the reliability of a trust model is also determined by its fault-tolerance. A trust model with greater fault-tolerance is considered more reliable.

Discussed below are some of the threats that a trust model needs to address in order to improve its reliability [Suryanarayana, Erenkrantz et al., 2004]. Table 1 (page 27) compares the various trust models with respect to these threats.

3.1.10.1 Impersonation

Impersonation refers to the threat posed by a malicious peer that portrays itself as another peer. The goal behind this threat could be to either misuse the privileges made available to the impersonated peer by other peers, or malign the impersonated peer through fraudulent interactions with other peers. As discussed earlier in Section 3.1.1, impersonation is typically addressed by (a) signing outgoing messages, and (b) verifying the identities of senders at the recipient's end.

3.1.10.2 Fraudulent Actions

In a peer-to-peer application, peers interact with each other in a variety of ways such as exchanging information, transacting deals, etc. While interacting with other peers in the system, a fraudulent peer may not completely fulfill its part of the transaction, or it may promise availability of certain services that it does not really offer. A trust model should (a) pre-interaction help peers identify such fraudulent peers, and (b) post-interaction enable peers to inform others about these fraudulent peers.

3.1.10.3 Mis-representation

A malicious peer may mis-represent the extent of trust it has in a victim peer and communicate these incorrect values to other peers. For example, a malicious peer could actually trust a victim peer but send out reports contrary to its knowledge. Depending upon the influence of the malicious peer, this may adversely affect the interaction of the victim peer with other peers in the system. Moreover, such a peer with malicious intentions could also mis-communicate the extent of trust another peer has in the victim peer. This problem is further compounded if the malicious peer acts as a forwarding relay between peers. Solutions to this problem include actively informing other peers about malicious peers and incorporating the opinions of multiple peers while making trust decisions in order to reduce the effect of mis-representation.

3.1.10.4 Collusion

Collusion refers to the threat posed when a group or groups of malicious peers actively try to subvert the system. Their actions may include spreading negative accounts of good peers and reporting greatly exaggerated positive accounts of other malicious peers in their clique. This leads to a situation where good peers are isolated and cannot decide whom to trust and may lead to a complete disruption of the system. Collusion can be addressed by encouraging good peers to actively (a) recognize groups of malicious peers and spread information about them, and (b) form robust groups themselves to counter the effects of collusion [Lee, Sherwood et al., 2003].

3.1.10.5 Addition of Unknowns

When a new peer joins an existing system, it does not possess trust-based knowledge about other peers in the system which may hinder it from interacting with other peers. Similarly existing peers in the system may tend to isolate the new peer since they lack trust information about the new peer. A trust model, therefore, should have a low barrier of entry for new peers so that new peers can easily participate in the system. Yet, at the same time, the trust model should provide sufficient measures to protect the system if the new peer turns out to be malicious. Addition of Unknowns also

encompasses the cold start problem which arises when the peer-to-peer system is first initialized and none of the peers have any trust information about any peer.

3.2 Trust and Reputation Technologies

As shown in Figure 1, we classify trust management into three categories: credential and policy-based trust management, reputation-based trust management, and social network-based trust management. This categorization is based upon the approach adopted to establish and evaluate trust relationships between peers.

In credential and policy-based trust management systems such as in [Blaze, Feigenbaum et al., 1996; Kagal, Cost et al., 2001; Yu, Winslett et al., 2001; Li, Mitchell et al., 2002; Yao, 2003], peers use credential verification to establish a trust relationship with other peers. The primary goal of such systems is to enable access control. Therefore their concept of trust management is limited to verifying credentials and restricting access to resources according to application-defined policies [Grandison and Sloman, 2000]. A resource-owner provides a requesting peer access to a restricted resource only if it can verify the credentials of the requesting peer either directly or through a web of trust [Khare, 1997]. This is useful by itself only for those applications that assume implicit trust in the resource owner. Since these policy-based access control trust mechanisms do not incorporate the need of the requesting peer to establish trust in the resource-owner, they by themselves do not provide a complete generic trust management solution for all decentralized applications.

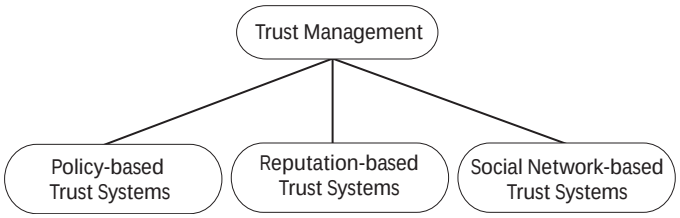


Figure 1: Trust Management Taxonomy

Reputation-based trust management systems on the other hand provide a mechanism by which a peer requesting a resource may evaluate its trust in the reliability of the resource and the peer providing the resource. Examples of such systems include SPORAS and HISTOS [Zacharia and Maes, 1999], XREP (section 3.2.2.2), NICE (section 3.2.2.4), DCRC/CORC [Gupta, Judge et al., 2003], Beta [Josang and Ismail, 2002], EigenTrust [Kamvar, Schlosser et al., 2003], etc. Peers in such systems establish trust relationships with other peers and assign trust values to these relationships [Zacharia and Maes, 2000]. Trust value assigned to a trust relationship is a function of the combination of the peer's global reputation and the evaluating peer's perception of that peer.

The third kind of trust management systems, in addition, utilize social relationships between peers when computing trust and reputation values. In particular, they analyze a social network which represents the relationships existing within a community and form conclusions about peers' reputations based on different aspects of the social network. Examples of such trust management systems include Regret (section 3.2.3.2) that

identifies groups using the social network, and NodeRanking (section 3.2.3.3) that identifies experts using the social network.

3.2.1 Policy-based Trust Systems

3.2.1.1 PolicyMaker

PolicyMaker [Blaze, Feigenbaum et al., 1999b] is a trust management system that facilitates the development of security features including privacy and authenticity for different kinds of network applications. Using PolicyMaker a peer may grant another peer access to its service if the providing peer can determine that the requesting peer's credentials satisfy the policies needed to access its service.

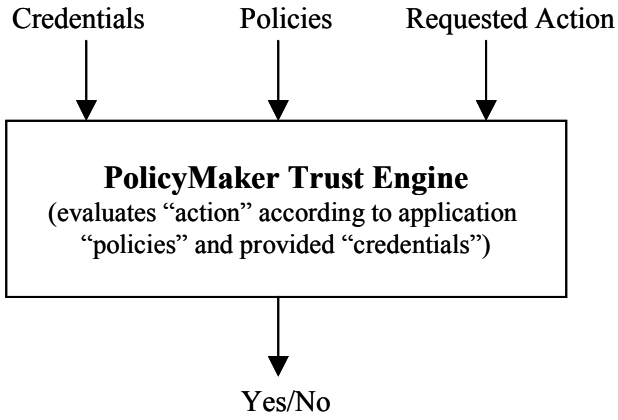


Figure 2: PolicyMaker Trust Engine

PolicyMaker's approach to trust management is based on the principles of unified mechanism, flexibility, local control, and separation of mechanism from policy. Unified mechanism refers to the ability of the trust management system to handle trust in a comprehensive manner by providing a common language for policies, credentials, and relationships. While flexibility refers to the ability of the system to support complex trust relationships, local control signifies whether a peer can make local decisions about the authenticity of credentials presented by other peers. The goal of separating the trust mechanism from the policies is to keep the authentication mechanism application-independent.

As shown in Figure 2, the PolicyMaker service acts like a database query service. A query is a request to determine whether a public key is permitted to perform a particular action according to a given policy. Queries are expressed in the **PolicyMaker Language** and contain a set of local policy statements, a collection of credentials, and proposed trusted actions. Security policies and credentials are defined in terms of predicates called filters that are associated with public keys. Filters accept or reject actions based on what the bearers of the corresponding public keys are trusted to do.

The PolicyMaker engine is responsible for evaluating these actions by interpreting the credentials with respect to the policy statements and returning a positive or negative

response. The PolicyMaker service can be used by applications as a library. In particular, we believe that it can be used in a peer-to-peer decentralized scenario by encapsulating the PolicyMaker engine as a module within each peer.

Trust management systems like KeyNote (RFC 2704) [Blaze, Feigenbaum et al., 1999a] and REFEREE [Chu, Feigenbaum et al., 1997] are based on the same principles as PolicyMaker. The main difference is that they place more responsibility on the trust engine. Unlike PolicyMaker which placed the task of credential verification upon the application itself, in KeyNote and REFEREE the trust engine is responsible for signature verification.

Certificate systems like PGP [Zimmermann, 1994] use certificates to verify that an identity actually belongs to a user, but do not verify whether a user is allowed access to resources. Access-based systems like PolicyMaker, instead, focus on verifying whether a particular key can be granted access to requested services. PolicyMaker also aims to make the trust management engine reusable by using policies and certificates that are based on predicates expressed in a general programming language. This enables the trust language for an application to change without any change to the trust management system. Similarly, trust descriptions are application-specific and require no changes to the trust management system. Furthermore, applications are responsible for describing trusted actions and taking appropriate actions based on correct descriptions leaving PolicyMaker to only ensure that described actions actually conform to the policies and certificates. These advantages have influenced the use of PolicyMaker as a trust management system in several applications [Blaze and Feigenbaum, 1997; Lacy, Synder et al., 1997].

PolicyMaker provides each peer with local control to specify its policies. However, it does not provide any mechanism to protect the anonymity of peers. Since peers do not depend upon trust information received from other peers, availability of trust information is not a critical requirement and so fault-tolerance does not pose a significant problem. Moreover, since peers do not query for and store trust information, bandwidth and storage costs are limited and do not affect its scalability. The main disadvantage of PolicyMaker is that it requires credentials and policies to be described in a particular language so that they can be processed by the PolicyMaker engine. While this makes it more reliable these constraints increases the complexity of the system.

3.2.2 Reputation-based Trust Systems

3.2.2.1 Trust-Recommendation Model

[Abdul-Rahman and Hailes, 1997] advocates an approach based on combining a distributed trust model with a recommendation protocol. The focus of the approach is on the following four goals – decentralization, trust generalization, explicit trust, and recommendations.

Decentralization allows each peer to take responsibility of its own trust policies and removes the need for those policies to be communicated to other peers. Thus it allows each peer to manage its own trust. Trust generalization is concerned with identifying that there are different dimensions to trust called trust categories, and trust in a peer

varies depending on these dimensions. Trust also needs to carry semantic meaning, so that trust values can be compared. Finally, in a large decentralized system, it may be impossible for a peer to have knowledge about all other peers. Therefore, in order to cope with uncertainty arising due to interaction with unknown peers, a peer has to rely on recommendations from known peers about these unknown peers.

In this model, a trust relationship is always between exactly two entities, is non-symmetrical, and is conditionally transitive. Mutual trust is represented as two distinct trust relationships. Two different types of trust relationships are distinguished. When one peer trusts another, it constitutes a direct trust relationship. But if a peer trusts another peer to give recommendations about another peer's trustworthiness, then there is a recommender trust relationship between the two [Abdul-Rahman and Hailes, 2000]. Trust relationships exist only within each peer's own database and hence there is no global centralized map of trust relationships. Corresponding to the two types of trust relationships, two types of data structures are maintained by each peer - one for direct trust experiences and another for recommender trust experiences. Recommender trust experiences are utilized for computing trust only when there are no direct trust experiences with a particular peer.

Trust categories are used by peers to classify trust towards other peers depending upon which aspect of that entity is under consideration. For example, a peer may trust another peer on a certain issue but may not trust it in another context. Similarly, since a peer may trust a certain peer more than other peers, comparable trust values are needed. A reputation is defined as a tuple consisting of a peer's name, the trust category and the specific trust value. A recommendation is defined as communicated trust information which contains reputation information.

This approach assumes the decentralized nature of entities as an integral part of its trust model. Each peer stores its own trust values and has autonomy over its own policies. Each peer also uses key-based encryption of messages so that recommendation information is not easily obtained by malicious peers. However, there is no provision to protect the identity of peers. The specific trust algorithm adopted makes several assumptions that trade accuracy for simplicity. This reduces the reliability of the trust model. Network traffic is caused primarily due to requests and responses for recommendations. Requests are sent only to trusted recommenders, and so bandwidth costs are limited. Though this approach essentially uses positive reputations, it also provides explicit reputation revocation to counter fraudulent actions of malicious peers. Additionally, the resistance to attacks of mis-representation and collusion is reactive and limited to decreasing the recommender trust of the concerned peers.

Each peer evaluates recommendations and computes reputations using data from its direct trust and recommender trust data structures. Once a peer has direct trust values for a certain peer, it does not evaluate recommendations for that same peer. Storage cost is determined by the size of direct trust and recommender trust values, and is dependent upon a peer's interaction behavior. Assuming that an increase in the size of the system results in a peer interacting with a greater number of peers, storage costs on each peer also tend to increase, potentially affecting scalability. Further, there are no mechanisms to ensure the availability of recommendation information in case a recommending peer gets disconnected or leaves the system abruptly. Having multiple

recommenders for every peer offsets this, but this is neither encouraged or enforced by the trust model and so the fault-tolerance ability of this model may be adversely affected. New peers that enter the system are equipped with an initial list of trusted peers with whom they can interact and slowly build up their reputation through good interactions.

3.2.2.2 XREP Reputation Protocol

The XREP [Damiani, di Vimercati et al., 2002] approach primarily focuses on P2P file-sharing applications. In addition to modeling the reputations of peers in the system, each peer also evaluates resources accessed from peers. Second, a distributed polling algorithm is used to allow these reputation values to be shared among peers, so that a peer requesting a resource can assess the reliability of the resource offered by a peer before using it. A similar approach has also been followed in the Poblano trust model [Chen and Yeager].

Each peer in the application is termed as a "servent" since it plays the role of both server and client by providing and accessing resources respectively. Each servent maintains information of its own experience on resources and other peers which it can share with other servents upon request. This information is stored in two repositories – a resource repository which associates a unique resource ID with a binary reputation value, and a servent repository which associates with each unique servent ID, the number of successful and unsuccessful downloads.

XREP is a distributed protocol that allows these reputation values to be maintained and shared among the servents. It consists of the following phases: resource searching, resource selection and vote polling, vote evaluation, best servent check, and resource downloading as illustrated in Figure 3. Resource searching is similar to that in Gnutella (section 4.2.1.1) and involves a servent broadcasting to all its neighbors a *Query* message containing search keywords. When a servent receives a *Query* message, it responds with a *QueryHit* message. In the next phase, upon receiving QueryHit messages, the originator selects the best matching resource among all possible resources offered. At this point, the originator polls other peers using a *Poll* message to enquire their opinion about the resource or the servent offering the resource. Upon receiving a Poll message, each peer may respond by communicating its votes on the resource and servents using a *PollReply* message. These messages help identify reliable resources from unreliable ones, and trustworthy servents from fraudulent ones.

In the third phase, the originator collects a set of votes on the queried resources and their corresponding servents. Then it begins a detailed checking process which includes verification of the authenticity of the *PollReply* messages, guarding against the effect of a group of malicious peers acting in tandem (collusion) by using cluster computation, and sending TrustVote messages to peers that request confirmation on the votes received from them. At the end of this checking process, based on the trust votes received, the peer may decide to download a particular resource. However, since multiple servents may be offering the same resource, the peer still needs to select a reliable servent. This is done in the fourth phase where the servent with the best reputation is contacted to check the fact that it exports the resource. Upon receiving a reply from the servent, the originator finally contacts the chosen servent and requests

the resource. It also updates its repositories with its opinion on the downloaded resource and the servent who offered it.

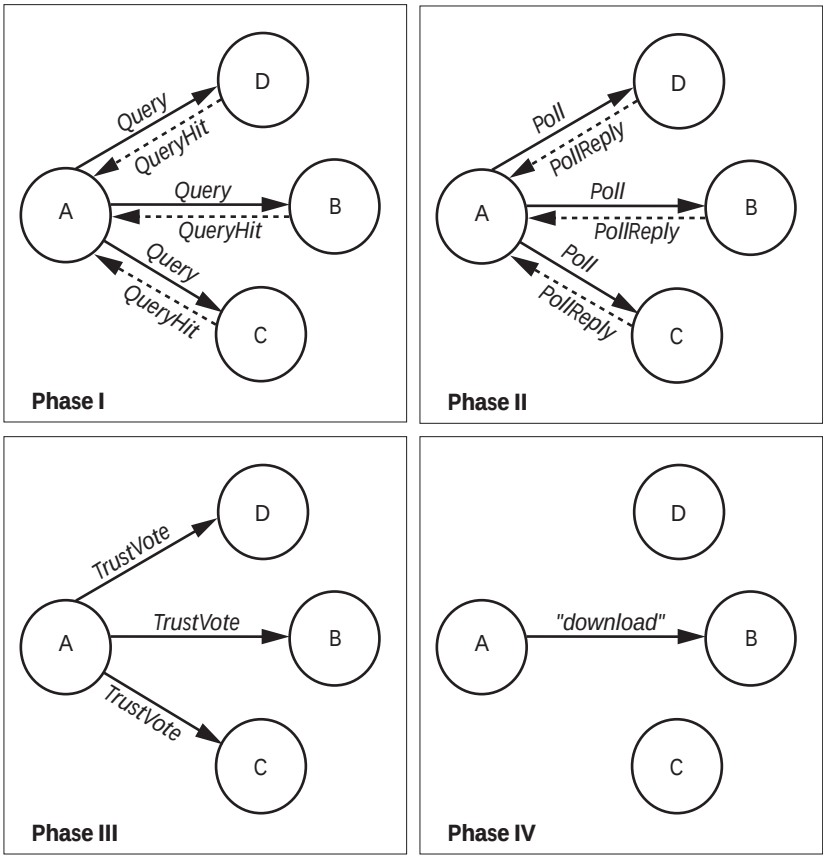


Figure 3: Phases of XREP

Each decentralized peer has local control over its trust information. Storage cost is due to the experience and servent repositories maintained by each peer and is limited to the direct interactions of the peer. However, the high bandwidth costs of the XREP trust model is its main shortcoming. This is due to two main reasons. The first is that *Poll* queries are broadcast throughout the network each time a peer needs to find out the reputation of a resource or a servent. This in turn affects the scalability of the trust model because an increase in the number of peers in the system can potentially lead to an exponential increase in the number of *Poll* queries and responses leading to possible network congestion. The second reason is the use of *TrustVote* messages that request confirmation of votes. There is also no provision to protect the anonymity of the peers in the system. Further, a mechanism that rates the referral ability of the peers is absent. Thus restricting mis-representation is limited to using opinions from multiple peers.

Further, in XREP *PollReply* messages are only sent in response to *Poll* queries and also only report a net reputation value of peers. Negative reputations are not explicitly distributed to peers. Yet, the XREP protocol provides certain measures to identify and recover from different kinds of malicious attacks such as shilling and pseudospoofing

[Damiani, di Vimercati et al., 2002] increasing its robustness. XREP assumes that resources are downloaded by multiple peers and hence a significant number of "correct" reputation values exist. Thus, the departure of peers from the system would not have an adverse effect. However, this assumption fails if good peers were to leave the system and colluding peers modify reputation values. In such cases, suitable measures are needed to ensure the availability of reputation data critical to counter these threats. The absence of these measures reduces the fault-tolerance capability of XREP. New peers joining the system can download well-reputed resources and subsequently offer them to increase their reputation.

3.2.2.3 P-Grid Trust Model

The P-Grid trust management approach focuses on an efficient data management technique to construct a scalable trust model for decentralized applications [Aberer and Despotovic, 2001]. This approach is motivated by the following reasons – firstly, a powerful trust model is worthless if it is not scalable; and secondly, data management becomes complex because data for computing trust cannot be obtained without computing the trust in the data sources, malicious peers can report misleading trust information, or trust data can be lost in traffic. To address these issues, the P-Grid approach divides the problem of decentralized trust management into three generic subproblems.

The first sub problem is to define a global trust model that determines whether a peer can be trusted or not. The second sub problem is to determine the local efficient computation that each peer needs to execute in order to approximately determine the trust in another peer. The last sub-problem is to study the effect of this local trust algorithm on the actions of malicious peers.

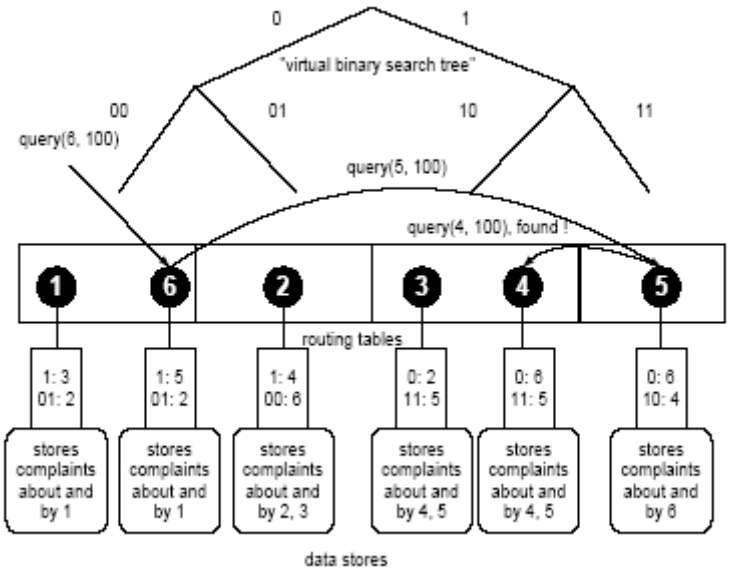


Figure 4: P-Grid data structure

The global trust model described is based on binary trust. Peers perform transactions and if a peer cheats in a transaction, it becomes untrustworthy from a global perspective. This information in the form of a complaint about dishonest behavior can be sent to other peers. Complaints are the only behavior data used in this trust model. Reputation of a peer is based on the global knowledge on complaints. While it is easy for a peer to have access to all information about its own interactions with other peers, in a decentralized scenario, it is very difficult for it to access all the complaints about other agents. This necessitates an efficient data storage model, called P-Grid [Aberer, 2001], to store trust data.

Figure 4 shows a typical P-Grid data structure. Peer 1's routing table contains an entry for peer 3 for paths starting with 1. So if a query starting with bit 1 reaches peer 1, it forwards it to peer 3. Peer 3 may address the query or forward it another peer depending upon the next sequence of bits in the query and so on. Similarly, when Query(6,100) reaches peer 6, it looks up its routing table and forwards the query to peer 5, since according to its routing table, all queries starting with 1 are addressed by peer 5. When Query(5,100) reaches peer 5, it takes the first two bits 10, looks up its routing table and forwards the query to peer 4 which answers the query.

Trust is computed by using P-Grid as storage structure for complaints. A peer can file a complaint about another peer and send it to other peers using *insert* messages. When a peer wants to evaluate the trustworthiness of another peer, it searches for complaints on it and identifies peers that store those complaints. Since these peers can be malicious their trustworthiness needs to be determined. In order to limit this process and to prevent the entire network from being explored, if similar trust information about a specific peer is received from a sufficient number of peers, no further checks are carried out.

The principal advantage of this approach is that it has an efficient way of storing and retrieving trust data and does not flood every peer in the system with queries about other peers, thus limiting storage and bandwidth costs. It is thus more scalable than approaches that broadcast trust queries to all peers in the system. The main disadvantage, however, is that a peer is forced to store data owned by other peers and does not have local control over the treatment of that data. Thus the system is not truly decentralized because peers have to implicitly agree to not alter data owned by others. It also does not employ any kind of mechanism to authenticate messages or explicitly protect the identity of peers.

The P-Grid trust model uses complaints to report behaviors of peers and so relies on a negative reputation-based scheme. Global trust information in the form of complaints is stored across peers. This leads to two problems. The first is that the entry and departure of peers from the system may result in important trust information being lost, resulting in a decrease of the fault-tolerance ability of the system. The second affects reliability since it is possible that a peer may end up storing complaints about itself which it may be motivated to alter or destroy. P-Grid addresses both these concerns by making trust data redundant across peers improving both fault-tolerance and reliability.

Peers can protect themselves against fraudulent actions by accessing complaints filed against fraudulent peers. Using trust data replicated across peers protects against the

possibility that a complaint is altered by a malicious peer. Mis-representation and collusion are further addressed by checking the trustworthiness of the peer that stores the complaint and the peer that reported the complaint, and combining opinions obtained from multiple trustworthy peers. The threat of addition of unknowns is addressed by trusting all new peers until complaints against them are reported.

3.2.2.4 NICE Trust Inference Model

NICE [Lee, Sherwood et al., 2003] is a platform for implementing distributed cooperative applications. Applications based on NICE barter local resources in exchange for access to remote resources. NICE provides three main services: resource advertisement and location, secure bartering and trading of resources, and distributed trust evaluation. The trust evaluation is necessary since malicious peers may threaten the reliable functioning of the cooperative system. Consequently, the objective of the trust inference model is to a) identify cooperative users so that they can form robust cooperative groups, and b) prevent malicious peers and clusters to critically affect the working of the cooperative groups.

NICE uses two trust mechanisms to protect the integrity of the cooperative groups - trust-based pricing and trust-based trading limits. In trust-based pricing, resources are priced according to mutually perceived trust. For example, if a peer A trusts B less than B trusts A, A will carry out a transaction with B only if B offers significantly more resources than A. Upon subsequent successful transactions with B, A will have more trust in B and so the difference in resources offered is less. In trust-based trading limits, instead of varying the price of the resource, the amount of the resources bartered is varied. This ensures that when transacting with a less trusted peer, a peer can set a bound on the amount of resources it loses.

Similar to some other trust models, the trust inference model utilizes the opinion of each transacting peer to rate the quality of the transaction. This opinion signed by a peer is called a cookie and is the measure of reputation in the NICE model. This cookie is stored on the other transacting peer which can use this cookie to prove its trustworthiness to other users. If, however, the opinion is negative, the peer storing it has no incentive to retain it, so in this case, the peer signing the opinion stores the cookies itself. Cookies are eventually expired or discarded so that constant storage space is achieved.

When a peer A wants to access B's resources, it sends B a set of credentials signed by B. Upon receiving this, B verifies that the cookies were indeed signed by it. Depending on the set of credentials, B may also decide to search for further references for A. These references along with the credentials are then used to compute the extent of B’s trust in A.

The trust inference algorithm can also be expressed using a directed graph called the trust graph (see Figure 5). In such a trust graph, each vertex corresponds to a peer in the system. A directed edge from peer A to peer B exists if and only if B holds a cookie signed by A which implies that at least one transaction occurred between them. The value of this edge signifies the extent of trust that A has in B and depends on the set of A’s cookies held by B. If, however, A and B were never involved in a

transaction and A wants to compute B's trust, it can infer a trust value for B by using directed paths that end at B.

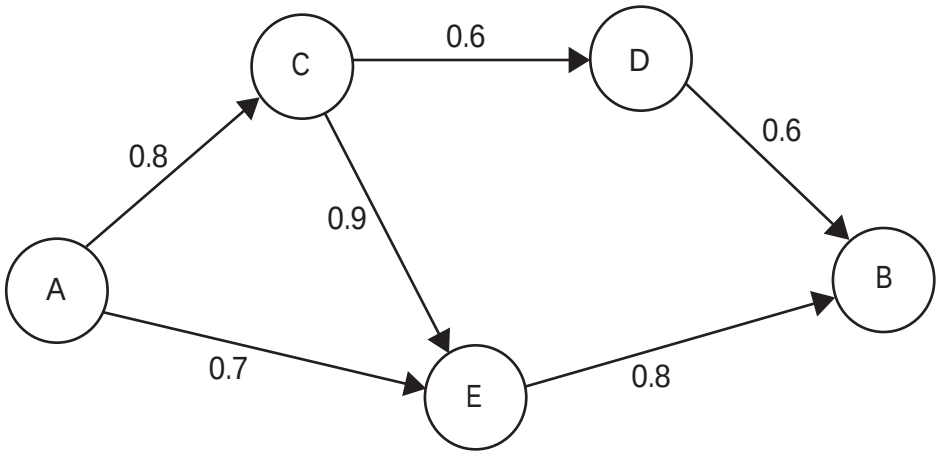


Figure 5: NICE trust graph
(Weights represent the extent of trust the source has in the sink)

Two trust inference mechanisms based on such a trust graph are described in the NICE approach. These are the strongest path mechanism and the weighted sum of strongest disjoint paths mechanism. In the strongest path mechanism, strength of a path can be computed either as the minimum valued edge along the path or the product of all edges along the path, and peer A can infer peer B's trust by using the minimum trust value on the strongest path between A and B. In the weighted sum of strongest disjoint paths, peer A can compute a trust value for B by computing the weighted sum of the strength of all of the strongest disjoint paths.

In other trust models, it is the responsibility of the resource owner, say peer B, to verify the trustworthiness of the requesting peer, say peer A, either using its own history of experiences or initiating a search for A's references through the peers it trusts. This is however subject to a potential denial-of-service attack since a malicious peer may continuously solicit other peers to verify its identity. The trust inference algorithm prevents this by putting the onus of acquiring the necessary credentials on A. Therefore, A has to search for B's cookies and present them to B if it wants to use B's resources. An additional advantage is that since each peer only stores cookies that explicitly benefit it, peers have an incentive to store cookies. This kind of incentive is absent in other trust models. However, an inherent flaw in allowing peers to store reputations about themselves is that they may tend to discard cookies that lower their reputations. As discussed already, in order to counter this, NICE allows a peer issuing a negative cookie to store the cookie itself.

Storing negative cookies and exchanging digests allows information about malicious peers to be dissipated to other peers in the system. This makes peers aware of fraudulent peers. When a peer initiates a search for negative cookies on a target peer, it only relies upon negative cookies received from trustworthy peers. Thus even if a malicious peer were to mis-represent its trust in the target peer, combining the opinions of other peers will help counter the mis-representation. There is no well-

defined solution to the problem of addition of unknowns. There are no cookies at the start of the system and peers build up reputation only with successful transactions.

One of the main contributions of the NICE approach is the ability of good peers to form groups and isolate malicious peers. To form such groups efficiently, peers maintain a preference list of potentially trustworthy peers that is constructed based on previous interactions and observations. This ability to form robust cooperative groups, along with the incentive to store cookies, improves the reliability of the system. NICE employs the use of both positive and negative cookies to achieve a more robust reputation scheme. NICE works in a purely decentralized fashion and each peer stores and controls data that benefits itself. Therefore storage costs even in the worst case are limited by the number of interactions with other peers. Further, to improve the efficiency of the cookie-search mechanism and limit bandwidth costs, NICE employs a probabilistic flooding-based search mechanism. These factors together improve its scalability.

Each cookie containing trust information about the requestor is signed by the owner and is verified by the owner upon receipt. Identifiers and public keys are used to verify the credentials of the requestor but there is no provision to protect the anonymity of peers. Since peers in the NICE trust model are responsible for storing reputation cookies themselves, their entry and departure from the network does not affect other peers significantly. While this argues for improved fault-tolerance, there are two exceptions. The first is when additional references required are unavailable because of the absence of peers. The second is when a peer storing a negative cookie departs the system encouraging the concerned malicious peer to continue with its activities.

3.2.2.5 XenoTrust

XenoTrust [Dragovic, Kotsovinos et al., 2003] is a distributed trust and reputation management architecture used in the XenoServer Open Platform [Dragovic, Hand et al., 2003] which is a public infrastructure for wide-area computing. As illustrated in Figure 6, the XenoServer Open Platform consists of three main entities: XenoServer, XenoCorp, and XenoServer Information Service. XenoServers provide services to clients. XenoCorp acts as a centralized certificate authority that is trusted by both servers and clients. The XenoServer Information Service is responsible for communicating state updates to the servers and uses a data structure called XenoStore to temporarily store updates. Since this platform is open and public, several security and trust mechanisms are needed. The purpose of XenoTrust is to model, administer, and distribute trust between participants (peers) in the XenoServer Open Platform.

There are two levels of trust in XenoTrust: authoritative trust and reputation-based trust. Authoritative trust is based on the verification of a peer's identity using the credentials issued by XenoCorp. More interesting is the reputation-based trust which is built through interaction between peers based on individual experiences. In order to accommodate newcomers to the system who have no initial experience with other partners, exchanging of reputation information between partners is advocated. But instead of storing the reputation vectors on each participant and using a pure peer-to-peer approach to facilitate this exchange of information, the reputation vectors are moved into XenoTrust itself. Thus, all information is aggregated in XenoTrust. This information is updated as new reputation information is received from peers.

A peer can update reputation values by sending a tuple consisting of reputation values and other relevant information to XenoTrust. This tuple is signed by the sender to prevent forgery. Peers access trust values using one of two different schemes. In the first scheme, relevant reputation changes are published by XenoTrust as notifications to subscribed peers. In the second scheme, peers employ a traditional polling "request/ reply" mechanism to query trust values.

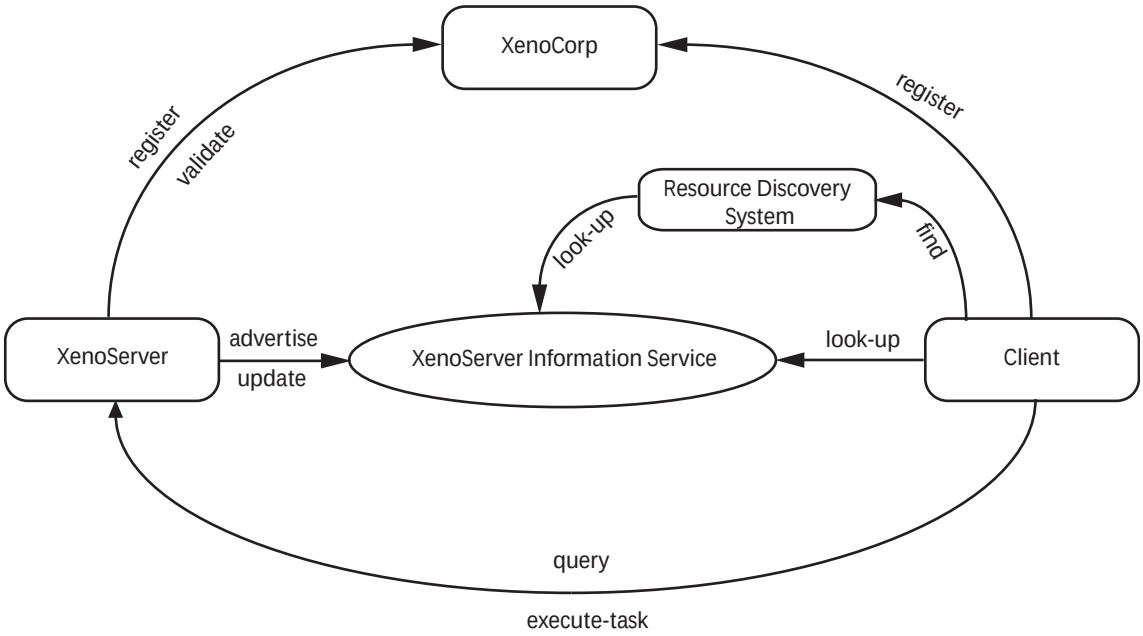


Figure 6: XenoServer Open Platform

The main disadvantage of XenoTrust is that it is not a pure decentralized system. Instead of distributing access and control over the trust data among the peers, XenoTrust acts like a trust server that aggregates and stores all the trust data on behalf of the peers. XenoTrust thus suffers from the typical shortcomings characteristic of any centralized system, including presenting a single-point-of-failure that can affect its reliability. It also does not allow peers the ability to maintain privacy of their views about reputations of other peers. Though XenoTrust's centralized architecture is not fault-tolerant, the departure of peers in XenoTrust does not affect the system significantly because no peer stores any data that can be lost. Other advantages include reduced bandwidth costs due to the use of publish/subscribe mechanism [Carzaniga, Rosenblum et al., 2001], minimal storage costs incurred by each peer, signing of tuples to prevent forgery, and a more scalable communication mechanism (publish/subscribe) than traditional polling schemes.

The reputation data reported by peers and stored in the XenoTrust trust server is used by the service to identify and remove fraudulent peers. This is done by carefully analyzing the reputation data obtained from multiple peers to protect against misrepresentation. Another example of misrepresentation is the case where a peer's reputation is positively exaggerated by those with similar interests. In order to counter this, peers combine opinions of multiple peers in the system. However, in the case of a collusion, this will work only if the number of correctly reported reputation values is

greater than the number of incorrectly reported reputation values. XenoTrust provides economic incentives to peers to report reputation data. This allows the aggregation of reputation data in the XenoTrust server and also provides new peers the ability to query for trust data. The new peers can thus identify and transact with trustworthy peers in the system.

3.2.3 Social Networks-based Trust Systems

3.2.3.1 Community-based Reputation

[Yu and Singh, 2000] was one of the first to explore the effect of social relationships of peers belonging to an online community on reputation in decentralized scenarios. It models an electronic community as a social network. Peers can have reputations for providing good service and referrals. Though the approach is focused towards multi-agent systems, it can be mapped to a peer-to-peer system by representing an agent as a peer. In such a system, peer agents assist users working with them in two ways. First, they help decide whether or how to respond to requests received from other peer agents in the system. And second, they help evaluate the services and referrals provided by other peers in order to enable the user to contact the referrals provided by the most reliable peer.

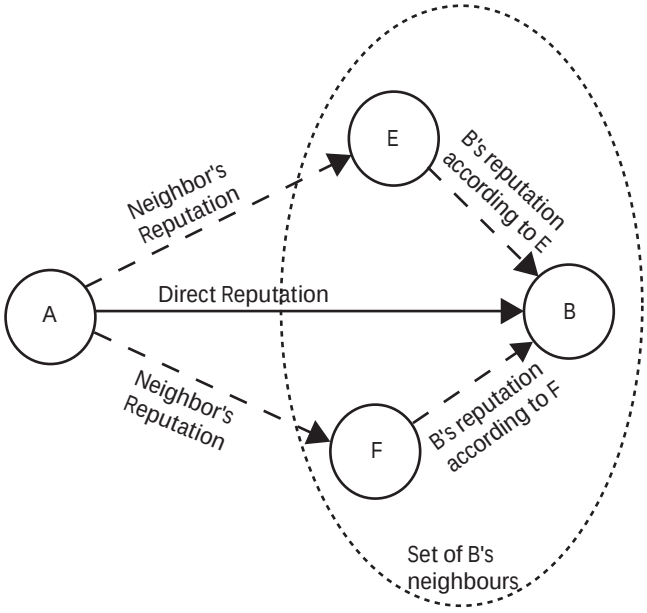


Figure 7: Computation of Reputation
(To compute trust in B, peer A relies on its direct interaction with B as well as relies on the referrals provided by B's neighbors E and F)

In this approach (see Figure 7), peer A assigns a rating to B based on its direct observation of B as well as the ratings of B as given by its neighbors, and A's ratings of those neighbors. When a user poses a query to its corresponding peer agent, the peer uses the social network to identify a set of potential neighboring peers who it believes has the expertise to answer that query. The query is then forwarded to this set of neighbors. A query sent to a peer contains three things – the question, the requestor

peer's ID and address, and a number specifying the upper bound on the number of referrals requested. When a query is received by a peer agent, it decides whether the query suits the user and if it should be shown to the user. The peer agent answers only if it is confident that its expertise matches the query. The peer may also respond with referrals to other trusted users who it believes has the necessary expertise to answer the query. Thus, a response may include an answer to the query, or a referral, or both, or neither.

When such a response is received by the original peer, it can use it in two ways. If the response contains an answer, it uses the answer to evaluate the expertise of the responding peer. This evaluation may result in a change in perspective about the expertise of the responding peer and of those peers who may have given a referral to the responding peer. This might result in a corresponding change in the social network. If the response contains a referral, the original peer can choose to follow it up.

Simulations with the above approach have shown in particular that the reputations of selfish and undesirable agent peers decrease rapidly, and the initial barrier of entry is low so that though a new peer in the system starts with zero reputation, it can increase its reputation steadily by cooperating with other peers. Another advantage of this model is that negative testimony about a malicious peer is quickly disseminated to other peers. This makes the system aware of fraudulent peers. The trust model relies upon trustworthy informant peers and the combination of multiple responses to address the problem of mis-representation. However, this technique can address collusion only if the good peers outnumber the bad peers. This limits the extent to which the trust model can counter collusion, and consequently affects its reliability.

The approach is completely decentralized and each peer maintains an interest vector, an expertise vector, and a set of expertise and reputation information of only some of its neighbors. Storage cost is therefore low and does not increase with the size of the system since only information about some neighbors is maintained. Since queries are only forwarded to neighbors with particular expertise, the communication mechanism is bandwidth efficient and increases the scalability. This approach encourages active propagation of reputation information to ensure that multiple peers contain redundant reputation information. This increases the fault-tolerance capability of the trust model. However, there is no provision for verifying query responses or for protecting the privacy of peers.

3.2.3.2 Regret

Regret [Sabater and Sierra, 2001] is similar in concept to TrustNet [Schillo, Funk et al., 2000] and includes the social dimension of peers and their opinions in its reputation model [Sabater and Sierra, 2002]. But rather than relying only on the corresponding social network as in TrustNet, Regret adopts the stance that the overall reputation of a peer is an aggregation of different pieces of information. Regret is based upon three dimensions of reputation - individual, social, and ontological. It combines these three dimensions to yield a single value of reputation. When a member peer depends only on its direct interaction with other members in the society to evaluate reputation, the peer uses the *individual dimension*.

If the peer also uses information about another peer provided by other members of the society it uses the *social dimension*. The social dimension is similar to the *regularity-based* trust described in [Minsky, 2003]. The social dimension relies on group relations. In particular, since a peer inherits the reputation of the group it belongs to, the group and relational information can be used to attain an initial understanding about the behavior of the peer when direct information is unavailable. Thus, there are three sources of information that help peer "A" decide the reputation of a peer "B" - the individual dimension between A and B, the information that A's group has about B called the Witness reputation, and the information that A's group has about B's group called the Neighborhood reputation. Figure 8 illustrates these various reputation relationships.

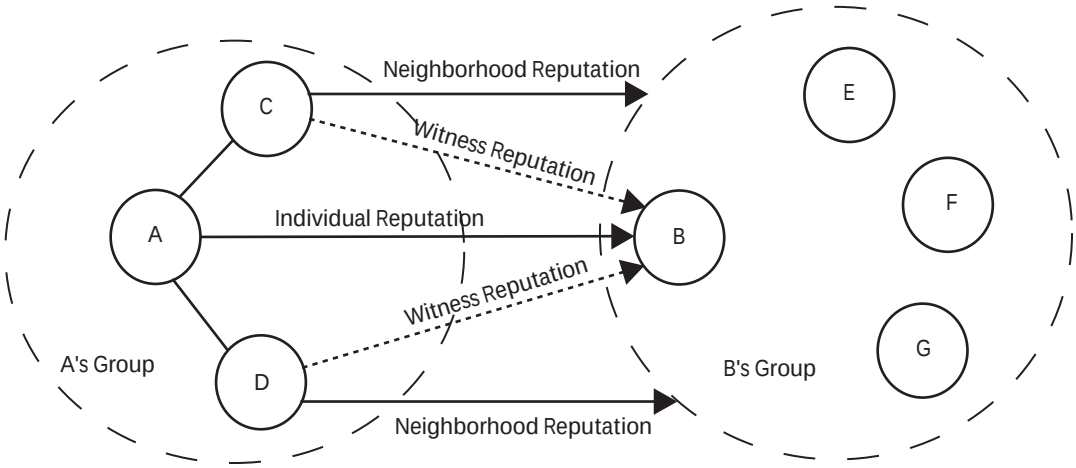


Figure 8: Individual and Social dimensions in Regret

Finally, Regret unlike TrustNet believes reputation to be multi-faceted and presents the following example as illustration - the reputation of being a good flying company summarizes the reputation of having good planes, the reputation of never losing luggage, and the reputation of serving good food. In turn, each of these reputations may summarize the reputations of other dependent factors. The different types of reputation and how they are combined to obtain new types of reputation is defined by the *ontological dimension*. Clearly, since reputation is subjective, each peer typically has a different ontological structure to combine reputations and has a different way to weigh the reputations when they are combined.

The Regret trust model is purely decentralized and allows each peer local control over its own data. It is a more complete trust model since it also considers group reputation and the ontological dimension while computing reputation. This increases Regret's flexibility and reliability. However, it is still limited in the sense that a peer does not cross group-boundaries to inquire peers from other groups about the reputation of a peer. Understandably if this were to be implemented, the reputation model would become quite complex and would require increased communication between peers. In addition, while the existing model is simple, each peer assumes an implicit trust in other peers belonging to the same group, thus exposing itself to malicious activity within its own group.

Regret expresses trust using both positive and negative levels of reputation. Since each peer stores group information in addition to peer information, additional storage space is required. If peers belonging to a group are considered neighbors, the Regret trust model reduces to the Community-based Reputation model discussed in Section 3.2.3.1 with dimensions, and therefore has similar bandwidth and scalability properties. The main shortcomings of the Regret model are the lack of credential verification, techniques to protect users' anonymity, and fault-tolerance mechanisms.

Upon detection of fraudulent actions, affected peers can modify not only the reputation value of the malicious peer but also that of the witnesses that recommended the peer. These changed values will forewarn other peers in the future. In addition to using this technique, a peer can combine opinions of multiple witnesses to detect misrepresentation. Like other trust models, collusion can be prevented by using the above social reputation mechanism as long as the number of good peers is sufficiently greater than the number of malicious peers. New peers that join the system start with zero reputation but quickly build up their reputation through successful interactions.

3.2.3.3 NodeRanking

NodeRanking, like TrustNet and Regret, utilizes social community aspects of peers to determine their reputation [Pujol, Sanguesa et al., 2002]. Most reputation mechanisms require frequent user involvement and feedback in order to be reliable and robust. The goal behind reputation systems like NodeRanking is to remove dependence upon the feedback received from other users, and instead explore other ways to determine reputation. NodeRanking views the system as a social network where each member peer has a position in the community. The location of a given member of a community in the network can be used to infer properties about the peer's degree of expertise or reputation. Members who are experts are well-known and can be easily identified as highly connected nodes in the social network graph. This information can be used by peers directly instead of having to resort to explicit ratings issued by each peer.

The NodeRanking algorithm helps create a ranking of reputation values of community members by using the corresponding social network. The reputation value assigned to a peer is based on the concept that each node on the graph has an associated degree of authority that can be seen as an importance measure. When the system first starts, it is assumed that all nodes have the same authority. The NodeRanking algorithm is then executed to calculate the authority values of all peers in the system. The social network can be considered as a directed graph where each edge has a direction. Edges that start from a node are called its out-edges and the nodes that they connect to are called its out-nodes. Similarly, edges that come into a node are called its in-edges and the nodes that they start from are called its in-nodes. The principal idea behind NodeRanking is that each node has an authority and a part of this authority is propagated to its out-nodes through its out-edges. Thus the authority of a node depends on the authority of its in-nodes.

The authority measure of a node is calculated as a function of the total measure of authority present in the network and the authority of the nodes pointing to it. Nodes that are not pointed to by any other node are assigned a positive default authority value. The resultant authority values obtained after executing the NodeRanking algorithm are used to infer the reputation of the peers in the community.

The principal shortcoming of NodeRanking is that it is centralized. While the NodeRanking algorithm does not require each peer to know about the rest of the system, the results from each peer are returned to a centralized node in order to construct the social network graph. This centralized node is then queried for reputation information by the peers. Naturally NodeRanking inherits the disadvantages inherent in any centralized scheme, namely, single-point-of-failure and issues of scalability. Additionally, there is no authentication of the communication between the centralized node and the peers and no mechanism to protect the privacy of peers. There is no scheme to prevent a malicious peer from mis-representing authority or to protect against a group of colluding malicious peers that may point to other peers in their own clique in order to increase their authority.

The measure of authority can be used to warn against fraudulent peers. If a peer is found to be committing fraudulent actions, such as sharing incorrect knowledge, peers in the system that currently point at it will start pointing at other peers, thus decreasing its reputation. Storage costs incurred by each peer are minimal. Network traffic is reduced but concentrated around the centralized node leading to possible congestion and delays in response. Interestingly, the departure of peers does not affect resource availability because no peer stores any reputation data. However, the social network graph and the corresponding reputations need to be recomputed to match the current system. Thus the departure of peers from the system increases computation overhead and is a threat to the system's reliability. Reliability is also affected by the single-point-of-failure. NodeRanking assigns a default positive authority value to a new peer so that it starts off with some positive reputation and its subsequent interactions (when other peers use its expertise and when it consults other peers) will determine its true authority.

Table 1: Comparison of Trust and Reputation Models against Threats

Technologies/ Properties	Policy-based Trust Systems	Reputation-based Trust Systems					Social Network-based Trust Systems		
	Policy-Maker	TrustRep	XREP	P-Grid	NICE	XenoTrust	Community-based Reputation	Regret	Node Ranking
Impersonation	Signature Verification	Signature Verification	Signature Verification	No	Signature Verification	Signature Verification	No	No	No
Fraudulent Actions	NA	Explicit reputation revocation	Use Vote Polling	Actively file complaints; Search complaints	Use negative cookies; Dissipates information about bad peers via digests	Reputation data reported is used by the service to eject malicious peers	Actively propagate negative ratings	Use "Social Reputation"	Less peers point to a fraud peer decreasing its authority
Mis-representation	NA	Limited to decreasing "Recommender Trust"	Combines responses from multiple peers; Referral trust absent	Compares to replicated trust data; Check trust of informers, combine their opinions	Use negative cookies from trusted peers	Combine data reported by multiple peers	Use reputation of neighbors; combine multiple responses	Use "Witness Reputation" and combine multiple responses	No
Collusion	NA	Limited to decreasing "Recommender Trust"	Use cluster computation, vote confirmation	Compares to replicated trust data; Check trust of informers, combine their opinions	Form robust cooperative groups using a preference list of peers	Vulnerable as it assumes Num of good peers > Num of bad peers	Works only if Num of good peers > Num of bad peers	Use "Social Reputation" works as long as Num of good peers > Num of bad peers	No
Addition of Unknowns	NA	New peers have an initial list of trusted peers with whom they interact	Providing well-known resources lowers barrier to entry	All new peers are trusted until complaints are reported	No cookies at start. Builds up trust with successful transactions	New peers query Xeno Trust for trust data; economic incentives to report reputation data	Zero at start, steady increase with successful interactions	Zero at start, rises quickly with successful interactions	Assigns a default positive authority value to a new node

Table 2: Comparison of Trust and Reputation Models

Technologies/ Properties	Policy-based Trust Systems	Reputation-based Trust Systems					Social Network-based Trust Systems		
	Policy-Maker	TrustRep	XREP	P-Grid	NICE	XenoTrust	Community-based Reputation	Regret	Node Ranking
Decentralized	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No
Local Control	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
Trust / Reputation Values	Binary	Discrete	Binary	Binary	Continuous	Continuous	Continuous	Continuous	Continuous
Type of Reputation	NA	Positive Negative	Positive	Negative	Positive Negative	Positive Negative	Positive Negative	Positive Negative	NA
Signature Verification	Yes	Yes	Yes	No	Yes	Yes	No	No	No
Anonymity	No	No	No	No	No	No	No	No	No
Bandwidth Cost	NA	*	*****	*	*	*	*	*	*
Storage Cost	NA	***	***	*	***	*	*	**	*
Fault-tolerance	*****	*	*	***	***	****	****	*	****
Scalability	*****	***	*	*****	*****	*****	*****	*****	*
Reliability	*****	*	***	****	*****	***	*****	***	**

4 Resource Discovery

4.1 P2P Discovery Properties

4.1.1 Local Control

Decentralized applications conceptually are supposed to consist of peers that are autonomous and have full local control over their data. However in certain cases, for example structured overlay networks, a peer may actually store and maintain data belonging to other peers. In order to prevent peers from changing the data that resides on them, peers in such systems are not enabled with local control over the data they maintain. Rather they have access and control to their own data residing on some other peer. We use the Local Control property to help distinguish systems that allow their peers local control over their data from those that don't.

4.1.2 Search Accuracy

Search Accuracy measures the effectiveness of a search mechanism. Since responses to a query may also include results that are not directly related to the query, it is important to distinguish them from exact matches. Therefore, assuming there exists at least one exact match for a query, Search Accuracy then represents the ratio of the number of matching results returned to the total number of results returned for that query. When an exact match is difficult to determine, Search Accuracy may also be evaluated based on the degree of relevance of the returned results to the query.

Clearly, there exists a relationship between Search Accuracy and the number of peers that are queried. As the number of peers queried increases, the probability that a match will be found also increases.

4.1.3 Search Flexibility

There are some search techniques that only allow resources to be searched by their names. The drawback of this is that users have to know the exact name of a resource in order to get a matching response. Richer search schemes on the contrary allow, for example, searches with keywords and thus provide far more flexibility to the user in specifying a query.

4.1.4 Performance

This property refers to the user-perceived latency of a search operation. In other words, it measures the time taken by a search operation to return a set of responses. While a centralized directory can pinpoint the exact location of a resource, a pure decentralized application may possibly have to initiate a system-wide search to determine the same. In addition to the time required for each peer to process queries, time is also required to set up connections to the next peer to whom the query is to be forwarded [Hong, 2001]. In such a case, the time required for a search operation will increase with the increase in the number of peers in the system. In general, replication and intelligent routing mechanisms help increase the performance of search operations.

4.1.5 Search Cost

In addition to reducing the number of messages transmitted to and fro between the peers, it is also necessary to reduce the number of peers that actually process the query. Gnutella-like systems are very inefficient and flood every query to all peers in the system. This results in every peer processing every request irrespective of whether it can satisfy a request or whether there are other peers that have already responded to the query. Ideally queries should be sent to only those peers that either have a matching resource or can point to the nearest one. The Search Cost parameter denotes the cost incurred by the peers to process a query and in this report is represented by the number of peers that process the query.

4.1.6 Bandwidth Cost

Bandwidth consumption in a P2P application is typically measured in terms of the number of messages exchanged between peers. Peers in a P2P system typically have different constraints regarding size and connectivity. Some peers with limited queue sizes may not be able to handle large number of messages. In addition if the system uses some flooding scheme, each peer is flooded with a lot of messages. This requires each peer to put in more effort to process messages and sometimes may also result in certain messages being dropped. These various reasons strongly affect the performance of the system. Therefore, reduction in bandwidth is one of the most important objectives of any search mechanism.

4.1.7 Storage Cost

Certain search schemes rely on some kind of routing information to intelligently route queries to peers that can better answer them. This entails peers to maintain routing tables and pointers to data and other peers in the system. The extra memory required to store this routing information is referred to as the storage cost. Storage cost also depends upon the number and size of resources stored upon a peer. For example, maintaining only one copy of a resource results in lower storage costs than maintaining multiple copies across peers.

4.1.8 Fault-Tolerance

One of the main characteristics of a P2P network is its constantly changing topology. This may be in addition or due to the transient nature of peers which may enter, leave or be disconnected from the system at any time. The system is expected to gracefully recover from the loss of data and routing information as peers enter or leave the network by mechanisms such as replication of data and routing along alternative paths respectively. It is also expected to continue providing connectivity and reliable service to all the peers without them being aware of any changes in the underlying network. Fault-tolerance denotes the extent to which a particular search mechanism can successfully cope with this challenge.

4.1.9 Scalability

Scalability refers to the ability of the search mechanism to support a greater number of peers or a greater number of interactions between the peers or both. For example, with increase in the number of peers in the system, it may be that each peer has to maintain larger indices. Advantages of this, however, include an increase in the probability of finding a resource, and a possible increase in resource redundancy across peers since more peers may be available to store multiple copies.

The ability of peers to handle multiple connections with other peers also affects scalability. For example, while a peer may be able to support parallel connections in a system with a small number of peers, it will be constrained to keep up the quality of its connections as their numbers increase.

4.1.10 Reliability

Reliability denotes the overall success of the search mechanism. It is hard to guarantee the reliability of search mechanisms in decentralized P2P systems because of two reasons. First, there is no centralized node that keeps track of the content of all the peers. As a result, resource requests may be routed in an arbitrary fashion and may never reach a resource owner. Second, resource availability is threatened because peers may depart from the system at any time. We combine these two reasons, namely, the effectiveness of the search mechanism and its fault-tolerance capability to decide its reliability.

4.2 P2P Discovery Technologies

We classify discovery technologies into two categories as illustrated in Figure 9: those that are used in unstructured networks and those that are used in structured networks.

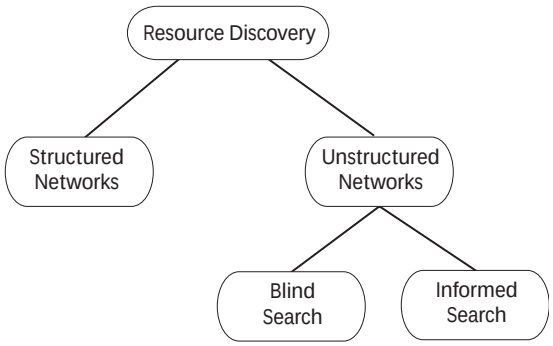


Figure 9: Resource Discovery Taxonomy

A structured network uses a distributed hash table (DHT) approach to create an overlay network that handles resource placement and ensures a bounded number of hops for every search query. Examples of technologies based on structured networks include CAN [Ratnasamy, Francis et al., 2001], Chord (section 4.2.3.2), Kademlia [Maymounkov and Mazieres, 2002], Koorde [Kaashoek and Karger, 2003], Kelips [Gupta, Birman et al., 2003]. These discovery technologies assume guaranteed peer and resource availability in the network and therefore do not work very well in environments where peers constantly leave and join the system. Therefore, these search mechanisms are typically geared towards specific architectures like ODISSEA [Suel, Mathur et al., 2003], OceanStore [Kubiatowicz, Bindel et al., 2000; Rhea, Eaton et al., 2003], PAST [Rowstron and Druschel, 2001b], and Bayeux [Zhuang, Zhao et al., 2001].

Discovery mechanisms in an unstructured network, on the other hand, do not assume any guarantees about the availability of peers and so do not put any restrictions on the topological placement of data. These systems assume the transient nature of peers in the system as an inherent characteristic of P2P networks. Examples of unstructured networks

include file sharing systems such as Gnutella (section 4.2.1.1). We further classify discovery mechanisms in unstructured networks into two categories: those that use variations of a blind search approach, and those that use different kinds of informed searches [Tsoumakos and Roussopoulos, 2003].

Blind search mechanisms employ non-intelligent flooding techniques to broadcast queries to peers in the system. Peers in such systems maintain no information about the network or the possible locations of resources to intelligently route queries to peers that can better handle them. Examples of such systems include those that use simple breadth-first search like Gnutella (section 4.2.1.1).

In informed search mechanisms, on the contrary, peers maintain some kind of routing information to forward queries intelligently to peers that can better handle the queries. This information may be based on either possible resource locations, or possible serving capacities of peers, or the number of successful responses to queries, etc. Informed search approaches offer smaller response time than blind search approaches but at the cost of increased overhead of maintaining intelligent routing tables. Examples of informed search mechanisms include Local Indices (section 4.2.2.3), Gia (section 4.2.2.4), etc.

It should be noted that the taxonomy based on structured and unstructured networks described above is not unique and has been suggested by others [Cohen and Shenker, 2002]. We believe this is mainly because discovery mechanisms have different requirements depending upon the underlying network. This classification therefore helps understand the guarantees that an application needs to make before using a particular discovery mechanism. Similarly, the sub-classification of unstructured networks-based mechanisms into those that use blind search techniques and informed search techniques is not unique and has been previously suggested by [Tsoumakos and Roussopoulos, 2003]. We believe this sub-classification helps distinguish applications that require increased system or network information from those that don't. For the purpose of our survey, we believe that these taxonomies are sufficient and promote a deeper study of the different needs of applications and the technologies best-suited for them.

4.2.1 Blind Search in Unstructured Networks

4.2.1.1 Gnutella

Gnutella [Gnutella, ; Kan, 2001] is one of the first peer-to-peer resource sharing applications that is purely decentralized. There is no single server or authority that regulates interaction among the peers, yet peers at the edge of the network can share resources with other peers. Peers communicate with each other through messages. Though there are a lot of interesting issues about Gnutella, we restrict ourselves here to only discussing its search mechanism.

Gnutella uses a breadth-first search (BFS) scheme to search for resources. When a peer needs to search for a resource, it broadcasts a search query message with an optional depth/hop count. Each message has a unique message ID called the UUID. When a neighboring peer receives this query, it stores this UUID and searches its contents to find out whether it has a match for the resource requested. If not, this query is progressively forwarded by the peer to all its neighbors until the specified depth/hop count is reached. If a peer contains a matching resource, it routes the response to the

peer from which it received the request using the UUID. This is followed by the actual resource exchange between the requesting peer and the resource owner through a direct connection like HTTP.

Though the search scheme used in Gnutella is faster than a depth-first search scheme, each peer broadcasts queries to all other peers resulting in unnecessary message traffic and computation overhead at each peer. This makes Gnutella quite inefficient and reduces its scalability. Moreover, search is limited only to file names. In the simplistic Gnutella model, there is no replication mechanism to ensure that multiple resource copies exist across peers. Therefore, the departure of a peer may result in non-availability of its resources decreasing the fault-tolerance capability and reliability of Gnutella. User-perceived latency is also significant since query responses are collected before being displayed to the user. However, querying a large number of peers potentially works towards improving the search accuracy. Gnutella's search mechanism offers peers a certain level of anonymity since a peer is aware only of its neighbors at any given time. This implies that every peer stores information about only a few peers in the system.

4.2.1.2 Iterative Deepening

Iterative Deepening [Yang and Garcia-Molina, 2002] involves initiating successive multiple breadth-first searches with increasing depth limits. Suppose there are to be three iterations, with the first iteration searching to a depth of 'a', the second to a depth 'b', and the third to depth 'c'. A policy that specifies the iterative deepening mechanism is then represented as P = {a, b, c}, and the time between successive iterations is represented by W.

Iterative Deepening with policy P as defined above works as follows. A source peer S first initiates a breadth-first search of depth 'a'. In turn, it receives response messages from peers. In addition to responding, when a peer at depth 'a' receives a query message, it stores the message temporarily and does not forward the query to other peers. If after waiting for time W the responses received satisfy the query peer S does nothing, else it starts the next iteration by initiating a breadth-first search with new depth 'b'. But instead of sending a new query, S uses a Resend message with a TTL (Time To Live) of value 'a'.

When a peer receives a Resend message it forwards the message to other peers unless it is at depth 'a', in which case it recognizes that the query has been resent, drops the Resend message and forwards the initial query temporarily stored with a new TTL value of 'b-a'. This ensures that peers that have already processed the query do not waste their resources in reprocessing. Similarly, the entire process is repeated with depth 'c' if none of the responses received satisfy the query.

While affording the same search flexibility as Gnutella, iterative deepening is more efficient than Gnutella's breadth-first search. While Gnutella also returns correct and relevant results, a lot of resources are wasted since the query is sent to every peer in the system. Instead, Iterative Deepening starts the search with a few peers and increases the span of search only in the following iterations. This improves its scalability. Another advantage of Iterative Deepening is that it aims for user satisfaction by providing as soon as possible an initial set of results, rather than

waiting to receive the responses of all the queries and then presenting the results to the user. The storage costs incurred are the same as in Gnutella since a peer is only aware of its neighbors at any time and thus only needs to store information about a few peers in the system. Iterative deepening does not have any scheme to replicate resources and therefore reflects the same fault-tolerance and reliability as Gnutella.

4.2.2 Informed Search in Unstructured Networks

4.2.2.1 Freenet

Freenet [Clarke, Sandberg et al., 2000; Langley, 2001] is a decentralized peer-to-peer information storage and retrieval system that allows users to share unused disk space. A resource (file) is identified with a unique location-independent key. Each peer maintains a local data storage which may not necessarily contain data owned by that peer. This data storage is made available to the network for reading and writing. Each peer also maintains a routing table that has a list of keys and the addresses of the peers that contain those keys.

Freenet uses a search scheme which is similar to depth first search (DFS) [Tarjan, 1972] but with intelligent routing. In this scheme, a peer forwards a query to only one neighbor at a time. A query is basically a request for a binary file key and also contains a hop count (TTL) attribute. Upon receiving this query, the neighbor searches its local data storage for the requested resource. If it does not find the requested resource, it looks up its routing table to locate the key that is nearest to the requested key and identifies the corresponding address. It then decrements the query's hop count and forwards it to this address. This process is repeated until either the resource is found or the hop count limit is reached.

When a peer finds a matching resource, it sends the resource back to the peer which forwarded the query. This peer will cache a copy of that resource in its own data store, create a new entry in its routing table associating that resource key with the resource provider and send the response upstream to the next peer and so on. Spreading caches of resources across peers helps in reducing the response time the next time the same resource is requested, and also increases the fault-tolerance and reliability of Freenet.

The principal advantage of this scheme is that since queries are routed intelligently using routing tables, other peers do not have to unnecessarily process queries, reducing both bandwidth and search costs and improving scalability. The storage cost incurred by a peer partly depends upon the size of the routing table maintained by it. Since the routing table does not list all the resources possessed by all the peers in the system, routing tables are smaller in size. Though this portends lower storage costs, its effect is offset because of caching which increases storage costs. Another advantage of Freenet is that search accuracy potentially increases because only exact matches are returned. The major drawback of Freenet is that the search scheme is sequential. However, the use of intelligent routing enables it to perform better than a Gnutella search. Searches for keywords are not supported in Freenet. Instead search is limited to querying for resources using binary file key.

4.2.2.2 Directed Breadth First Search (DBFS)

In the Directed Breadth-First Search mechanism [Yang and Garcia-Molina, 2002], the originator sends query messages to only a subset of its neighboring peers. This subset of peers is selected based on how well they are connected to peers providing higher quality of resources so that the results returned are consequently of higher quality. To intelligently choose such a subset of neighboring peers, every peer maintains relevant data about each of its neighbors. This may include the number of results received through that neighbor for past queries, the quality of the connection with that neighbor, the number of responses that required the least number of hops, the bandwidth and storage capacities, etc.

The advantage of the DBFS mechanism is that response time to a query is reduced since it directs the search queries towards peers that are more likely to respond with positive results. Another positive side-effect of this is that lesser number of peers are queried, significantly reducing search and bandwidth costs and increasing scalability. However, storage costs are increased since each peer has to keep track of extra routing-relevant information about each of its neighbors. Without any provision for data redundancy, fault-tolerance and reliability of the system is reduced. Search accuracy and flexibility is the same as in Gnutella.

4.2.2.3 Local Indices

In the Local Indices mechanism [Yang and Garcia-Molina, 2002], each peer maintains a local index of the content of all neighboring peers within a hop distance of 'r'. This hop distance is called the radius of the index and for value 0 reduces to the case of a breadth-first search where each peer indexes only its own content. Maintaining indices is useful because a peer can process a received query on behalf of every node within 'r' hops. This helps in reducing the number of peers that actually process a query, and user-perceived latency.

In the Local Indices Search mechanism a policy P specifies one or more depths at which a query should be processed. All peers at depths not specified in the policy do not process the queries and instead only forward them to peers at the next depth. For example, if P = {1,5}, a query will be processed only by peers at depth 1 and forwarded to peers at depth 2 who will forward the query to other peers without processing it. Only peers at depth 5 will ultimately process the query.

Since this search mechanism relies on accurate indices to be maintained at each peer, it is important to update these indices when a peer joins or leaves or changes its content. In particular, a new peer uses a Join message to reach out to other peers in the system. This is followed by an exchange of metadata about content between the existing peers and the newly joined peer. On the other hand, when a peer leaves the network or is cut off from the rest of the system, peers that have indexed its content remove its corresponding metadata after an appropriate time period. In both these cases as well as when a peer updates its content, it sends out an Update message with a TTL of 'r' so that its neighbors can also update their indexes to reflect the changes.

This approach has been adopted as part of several discovery mechanisms (see section 4.2.2.4 and 4.2.2.5). Advantages of this approach include reduced search and bandwidth costs, and faster searches. These in turn improve its scalability. However

storage costs are higher than in Gnutella because each peer now stores an index of the content of its neighbors. But this storage cost does not increase significantly with the increase in the number of peers since the number of neighbors a peer has will remain a fraction of the size of the system. Thus increased storage cost has no adverse effect on scalability. Using metadata to search for resources and find matches increases the search flexibility. There is no data replication or any other mechanism to deal with the departure of peers from the system. Consequently, fault-tolerance and reliability are low.

4.2.2.4 Gia

Gia [Chawathe, Ratnasamy et al., 2003] is a P2P file-sharing system like Gnutella but with modifications to Gnutella's design to achieve better scalability. In particular, the contributions of Gia are mechanisms for the dynamic adaptation of the network overlay and improved search algorithms. Gia is designed specifically for unstructured networks where the system topology and data placement is not controlled. However it has been shown to serve systems three to five orders of magnitude better than that of Gnutella.

Gia uses random walks to search for resources instead of flooding the entire network with messages. Since the changing topology of the P2P network is a significant factor that affects the random walk mechanism, a dynamic topology adaptation algorithm is included. This algorithm guarantees that peers that can handle a large number of queries (high-capacity peers) are highly-connected so that more queries are routed their way, and that most peers are within a short network distance of such high-capacity peers. This reduced network distance ensures that search accuracy is improved since peers with required resources are a few hops away from the high-capacity peers.

Gia introduces a token-based flow control algorithm to prevent the overloading of peers. Each peer periodically assigns flow-control tokens to its neighbors. A flow-control token represents a single query that the peer is willing to accept. So a neighboring peer can send a query to a peer only if the neighbor had received a token from this peer. A peer can decrease the rate at which it allocates tokens to its neighbors if it receives more queries than it can handle. Similar to the Local Indices approach in 5.2.3, all peers store pointers to the content offered by their immediate neighbors. This in combination with the dynamic adaptation algorithm ensures that the highly connected peers have higher capacity so that they can handle more queries.

Though random walk search scheme qualifies as a blind search mechanism, the Gia random walk search protocol is different and is biased towards high-capacity peers [Lv, Cao et al., 2002]. Instead of forwarding a query to a randomly chosen neighbor, a peer forwards the query to the neighbor for which it has the greatest number of flow-control tokens. To avoid redundant paths, each query is assigned a unique ID by its originator. Each peer keeps track of the neighbors to which it forwarded a given query. If a peer receives the same query again, it forwards it to a different neighbor. A query can specify the maximum number of responses it expects which is decremented every time a peer finds a matching response for that query. If a peer itself contains the requested resources or has links to its neighbors with that resource, before forwarding the query the peer appends the addresses of the peers that contain that resource. This

restricts a neighboring peer that owns a matching resource from responding if it finds itself already listed in the query message. Query responses are sent back to the originator using the reverse path of the query.

Gia enables keyword searching and adapts its protocols to account for capacity and bandwidth constraints that peers may have. Enforcing peers to maintain and share indices of their neighbors' content improves the search efficiency and scalability at the expense of storage overhead on the peers. Other advantages include bandwidth-efficient communication among peers, and reduced search costs. Multiple high-capacity peers avoid the problem of a single-point-of-failure. All these factors combine to make Gia a more reliable search mechanism than Gnutella. However, while Gia also implements mechanisms such as query resilience that enable the system to recover from the effects of sudden departures and joins of peers, there are no data replication mechanisms to ensure resource availability.

4.2.2.5 Probabilistic Search Protocol

The Probabilistic Search Protocol as described in [Menasce and Kanchanapalli, 2002] attempts to address the content location problem using a probabilistic approach. Whereas the deterministic location approach aims to locate a set of peers that manage a resource given the name of the resource, the probabilistic location approach aims to locate with a given probability a set of peers given a resource name. The motivation in introducing a probabilistic measure is because it is inefficient to search all the peers in the system for a particular resource. Therefore, the probabilistic resource location protocol trades performance and scalability for the probability Pf that a resource is located. The protocol can be altered to get the desired value of Pf. Clearly, the ideal objective would be to achieve Pf = 1 while reducing the consumption of system resources like bandwidth, number of peers searched, and storage.

Each peer in the system has a Local Directory that points to its resource, and a Directory Cache that maintains a mapping between a resource and its possible location. A resource can be identified with a unique global identifier (GUID). Each peer has a local neighborhood comprising of a set of peers that are within certain number of hops from that peer. Two messages SearchRequest and ResourceFound are used to encapsulate query requests and responses respectively. The SearchRequest also specifies a TTL (time to live) parameter that dictates the depth of the query.

Upon receiving a SearchRequest message, a peer first searches its Local Directory and then its Directory Cache. If it finds the queried resource in the Local Directory, it responds with a ResourceFound message which updates the Directory Cache at every peer it visits along the reverse path back to the originator. On the other hand, if the resource is found in the Directory Cache, the SearchRequest message is forwarded to the corresponding peer. If, however, the resource is not found in either the Local Directory or the Directory Cache, the peer broadcasts the SearchRequest message to each peer in its neighborhood with a probability called the broadcast probability.

In this approach, typically a peer only forwards queries to peers based on their indices maintained in its Local Directory. In other words, only a few number of peers actually process queries leading to a reduction in search costs. Experiments done with this search mechanism [Tsoumakos and Roussopoulos, 2003] indicate that query flooding

is frequent only at system-start. Subsequent query flooding are limited to a small depth since more number of nodes have pointers to resources. Therefore, after system start only a small number of messages are needed to find responses to queries, thus reducing bandwidth consumption. All these factors tend to improve the scalability of the PSP approach. However, though using a directory cache that is updated with every query response helps better the search speed and search accuracy, it costs each peer extra storage space to maintain the directory cache. With increase in the number of peers, the size of the directory cache also increases across every peer limiting the scalability of the approach. Search flexibility of the PSP scheme is not clearly known. Further, there is no data replication mechanism to ensure resource availability in the event of peer departures. Consequently, fault-tolerance and reliability are affected.

4.2.3 Structured Networks

4.2.3.1 pSearch

pSearch is a decentralized non-flooding P2P information retrieval system [Tang, Xu et al., 2003]. The main approach here is to adopt some rationale instead of randomly allocating documents among distributed peers. It uses a logical network called a semantic overlay in which resources are distributed based on their semantics such that the network distance between two resources is proportional to the difference in their semantics.

pSearch uses Latent Semantic Indexing [Deerwester, Dumais et al., 1990] to generate semantics of documents/resources [Tang, Xu et al., 2002]. These are used to distribute document indices throughout the P2P network based on a content addressable network (CAN) [Ratnasamy, Francis et al., 2001] which provides a distributed hash table abstraction over a cartesian space. The semantic overlay is thus mapped to the physical nodes in the CAN. In particular, the semantic vector of a document is used as the key to store the document index in the CAN. Depending on its semantics, each document can be considered as a point in the cartesian space. Two documents having similar content will naturally be positioned close to each other in this space. A query for a resource can also be positioned similarly in this space and all documents in a particular range of the query would be considered good matches.

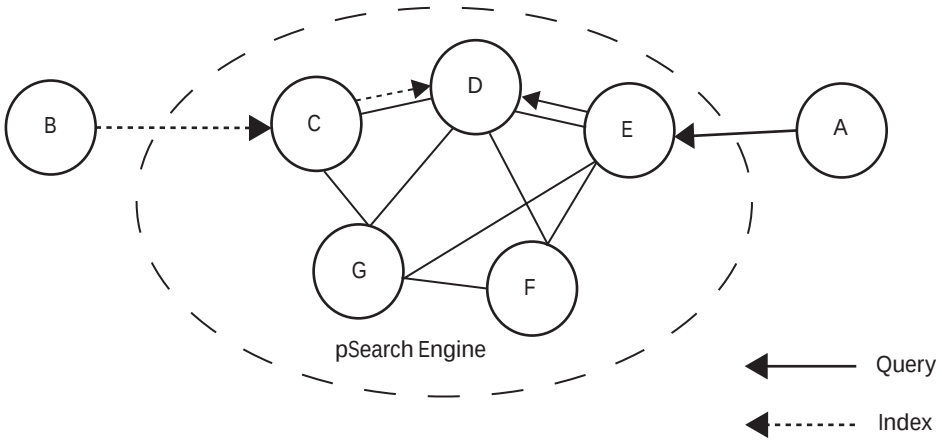


Figure 10: The pSearch system

In pSearch, peers with good connectivity collectively form a pSearch Engine (see Figure 10). All peers within the pSearch Engine are exactly similar in functionality and help build and maintain document indices, and also respond to queries submitted by peers outside the Engine. It is possible that the network may be imbalanced because the semantic vectors generated are not distributed uniformly in semantic space. To counter this, pSearch partitions the semantic space along more dimensions to help distribute the indices more uniformly across the peers. Furthermore, when the load inside the Engine is high, more peers can join the Engine.

pSearch has an enhanced search mechanism by allowing resources to be searched on the basis of their semantics. It reduces the number of nodes accessed by using appropriate index distribution and feedback information received from recently processed queries. In addition, since indices of semantically related documents tend to be co-located in the pSearch network, a lesser number of nodes are searched reducing search and bandwidth costs. Since queries are always directed towards a matching resource, two benefits are achieved. The first is that it decreases the time required for searches, and the second is that the search accuracy is improved since a returned document will be the closest (or overlapping) in semantic space to the requested document. In fact, pSearch has been found to achieve performance comparable to centralized information retrieval systems by searching only a small number of nodes. Efficient document distribution, and decreased make pSearch more scalable than other search mechanisms.

The structured network underlying pSearch increases the impact caused by the entry and departure of peers from the system. To counter this, pSearch enforces replication of both content and indices stored on an Engine peer on some of its neighbors. This increases its fault-tolerance and also contributes towards increasing its reliability. However, this also increases storage costs but since only the content of neighbors is replicated on a peer, storage cost is controlled and does not affect scalability significantly. The main disadvantage of pSearch is that it is not strictly decentralized. This is because resources are distributed according to a system-wide algorithm; as a result, a peer may not have actual local control of its own resources and they may even be located on a distant peer.

4.2.3.2 Chord

Chord [Stoica, Morris et al., 2001] is a distributed lookup protocol that efficiently allocates resources to peers. Chord basically performs just one operation - given a key, it maps the key onto a peer which may store associated resources, for example, files, documents, etc. depending upon the nature of the application. Thus a key corresponds to a resource or service that a peer can provide. To map keys onto peers, Chord uses consistent hashing [Karger, Lehman et al., 1997] that balances the load on each peer by roughly assigning all peers an equal number of keys. The consistent hash function assigns each peer and key m-bit identifiers by hashing the peer's IP address and the key respectively. These identifiers are ordered around an identifier circle modulo 2m also called the Chord ring as shown in Figure 11. A key is assigned to the first peer whose identifier is equal to or follows the identifier of the key. This peer is called the successor node of the key, and is the first peer clockwise from the key in the Chord ring.

In an open decentralized system where peers can leave the system at any time, consistent hashing also limits the number of keys that need to moved from one leaving peer to another peer in the system. When a peer 'p' joins the system, certain keys previously assigned to p's successor are now assigned to p. Similarly, when peer p leaves the system, all of its assigned keys are reassigned to p's successor. Therefore, there is minimum disruption when peers enter and leave the system.

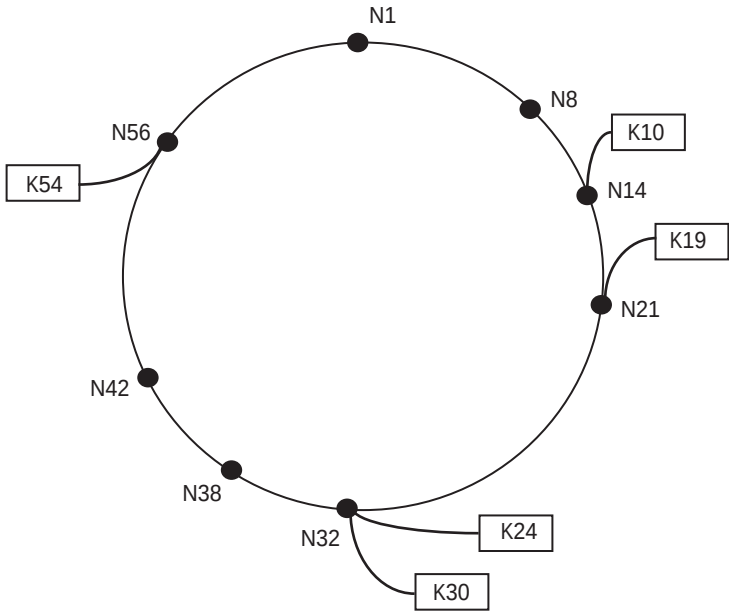


Figure 11: Chord Ring with 8 nodes storing 5 keys

Chord improves the scalability of consistent hashing by not requiring each peer to know about every other peer. Rather, each peer only needs to know how to contact its current successor node in the Chord ring (see Figure 11). Queries for a given key are passed around the ring via these successor pointers until the peer that maps to the key is encountered. However, this process can be slow. Therefore, in order to accelerate lookups, each peer maintains additional routing information in the form of a finger table. A peer p's finger table contains only 'm' entries, with the i^{th} entry containing the identity of the first peer that succeeds p by at least 2i-1 on the Chord ring. This entry includes both the Chord identifier and the IP address of the relevant peer. Thus each peer stores information about only a small number of peers that helps it to determine approximately the location of a key. Each Chord peer also runs a stabilization protocol periodically to keep the finger tables and successor pointers updated. This protocol learns about newly joined nodes and determines whether they should be its predecessor or successor peers in the Chord ring.

Chord provides an equal distribution of keys (resources) among peers and facilitates faster and efficient lookups because of finger tables. Moreover, search and bandwidth costs are lower since queries are not flooded to all the peers in the system. All these factors combine to make Chord extremely scalable. However, like in pSearch, a peer does not have local control over the resources it owns. Thus a Chord-based system is

not decentralized in the truest sense. Using hash values to search for resources also limits the search flexibility.

To address concurrent departure and arrival of peers, each peer in Chord additionally maintains a list of successor peers. Thus, if a peer's successor in the Chord ring suddenly leaves the system, the peer can substitute the successor with another one from the list. Peers voluntarily leaving the system can also transfer their keys to their successors to ensure resource availability. However, since resources are not actively replicated across peers, resource availability is affected in the face of sudden involuntary departure of peers from the system. This reduces the reliability of Chord to a certain extent.

Pastry [Rowstron and Druschel, 2001a] and Tapestry [Zhao, Kubiatowicz et al., 2001] are two routing and location mechanisms that are very similar to Chord. The main difference is that instead of forwarding messages based on numerical difference between the source and destination addresses as Chord does, Pastry and Tapestry utilize prefix and suffix based routing mechanisms respectively [Zhao, Huang et al., 2003]. Further, unlike Chord, they utilize a natural correlation between overlay logical distance and network distance in the underlying network which ensures that queries never travel long physical distances for every close logical hop.

4.2.3.3 Gridella

Gridella [Aberer, Punceva et al., 2002] aims to address the inefficient data access problems of P2P applications like Gnutella, using a decentralized scalable data access structure called the P-Grid [Aberer, 2001]. Each Gridella peer internally has a client-server architecture. While the Gridella client component provides a user interface, the Gridella server component handles data management and communication with other peers.

P-Grid is a virtual binary search tree that distributes replication over peers in the network and uses randomized algorithms for search and access. Search keys are represented in a binary form and distributed across peers. Each peer stores part of the overall tree. Each peer's position is determined by its path in the form of a binary bit-string that represents the subset of the tree's overall information that the peer is responsible for. To achieve redundancy in order to account for peers that may be offline, multiple peers can be responsible for the same path.

P-Grid provides a mapping scheme to compute a binary representation from a filename string. A requesting peer may send a query in the form of binary bits to any peer in the system. Upon receiving a query, the receiving peer checks if it has the corresponding resource. If it cannot address the query, it searches its routing table for the longest common prefix and forwards the query to the corresponding peer. This process is repeated with subsequent bits in the query until the query reaches a peer that can exactly match the query. This peer returns a reference to the associated data to the query originator which can then request the data specifically.

For each bit in its path, a peer stores the address of at least one other peer that is responsible for the other side of the binary tree at that level. This is necessary so that any query that cannot be satisfied can be forwarded to a peer whose path reveals that it

can better address the query. The P-Grid construction algorithm ensures that routing tables of peers always provide at least one path from the peer receiving a request to a peer holding a replica of the requested resource.

The main advantage of Gridella is that search for resources is bandwidth efficient since queries are not flooded to all the peers, and faster since there are multiple copies of resources and at every step the query is routed to a peer that is closer to the result. These make Gridella quite scalable. However, maintaining multiple copies can also potentially decrease the search accuracy. In particular, when resources are to be updated, the updating peer uses a limited breadth-first search to identify all the replicas and replace them with fresh copies. This limited breadth-first search does not ensure that all copies are found; as a result, the probability of finding a fresh resource decreases. Gridella addresses this concern by initiating multiple searches on the P-Grid and choosing the correct answer based on the highest number of identical answers. Though this increases its search effectiveness and reliability, it also increases search and bandwidth costs.

P-Grid facilitates search of resources using binary representation of only resource names. The search mechanism is therefore limited in flexibility. P-Grid handles the departure of peers from the system by making data redundant across multiple peers. This, in addition to every peer maintaining a part of the P-Grid tree as well as a routing table to route queries to suitable peers, increases storage costs considerably. This, in turn affects the scalability of Gridella. A peer may also not have local control since it may not store the resources it owns and therefore, Gridella is not a pure decentralized system.

4.2.4 Other Search Mechanisms

4.2.4.1 A Mobile agent-based solution

[Dunne, 2001] describes a mobile agent based solution to address the discovery problem in P2P applications. The main focus of this approach is to reduce the bandwidth consumption that results when peers establish channels of interaction using mobile agents [Ghezzi and Vigna, 1997]. Mobile agents can help encapsulate these interactions along with any data required and can be sent as a discrete piece of network traffic from one peer to another. Therefore when a mobile agent arrives at the destination peer, all ensuing interactions are limited and localized to the peer, thus reducing network traffic significantly.

When a peer wants to participate in a network, it creates a mobile agent. The created agent updates itself with all information about its creator peer. Two parameters of the mobile agent are set that control the breadth and depth of the search. These are the journey time that denotes the maximum number of peers that may be visited before the agent returns to the origin peer, and a branching factor that determines the number of times the agent may be cloned at any peer. Initially the agent is given the addresses of a few other peers in the network. It then clones itself so that an agent can be sent to each of these peers. After arriving at each peer, it exchanges resource information with that peer. Thus the peer comes to know of other resources in the system, and the mobile agent updates itself with the peer's resource information.

If the agent's journey time expires at a peer, it returns to its creator peer and updates it with all the information it has collected. Else, it clones itself and sends a clone to each peer that the current peer knows and that was not already visited.

This approach provides several advantages. Mobile agents are autonomous and can add dynamic information about the network to their knowledge base as they progress through it. This increase in knowledge facilitates increase in search accuracy. Further, they can also be cloned so that they may function in parallel. This ensures that network resource discovery is completed sooner. If there is a shortage in the network resources, search accuracy can be traded for efficiency by decreasing the journey time and branching factor accordingly. If a peer's resources are critically limited, it can refuse to accept mobile agents. Another advantage is that even if some mobile agents are destroyed, other agents still have a positive impact.

Peers have local control over their resources, but also store data about all known peers. Storage costs, therefore increase linearly with the increase in the size of the system and can potentially reduce scalability. Though the approach does not shed any light on the nature of the resource information obtained through the mobile agents, we believe that this mechanism can afford flexible search mechanisms by employing mobile agents with suitable capabilities. One significant aspect of this search approach is that every peer creates mobile agents to collect and update resource data across the network, and maintains that data irrespective of whether those corresponding resources are really required. Though this facilitates faster search responses, it can be wasteful if the resource is not really needed. This approach also lacks suitable mechanisms such as data replication to alleviate the effects of departure of peers from the system. This can potentially reduce its reliability. However, this is difficult to determine since this approach lacks evaluation in a large real-world system.

4.2.4.2 PlanetP

PlanetP [Cuenca-Acuna, Peery et al., 2001] is an infrastructure that allows users to set up information sharing P2P communities without the presence of any centralized server. Each peer creates an inverted (word-to-document) index of the documents that it wishes to share, summarize this index in a compact form, and diffuse the summary throughout the community. Using these summaries, any peer can query against and retrieve matching information from other peers in the system. The goal is to provide a rich search mechanism while at the same time improving search efficiency. A similar approach has been used in other search mechanisms for example, YouSearch [Bawa, Bayardo Jr. et al., 2003].

In Planet P, data is represented using XML. The XML document can also contain links to external files. A peer can share an XML document with other peers by choosing PlanetP to index the document. Since an index can be huge, PlanetP utilizes a Bloom filter [Bloom, 1970] to summarize the index. This condensed index is then distributed to other peers using a variation of a rumor-mongering algorithm [Demers, Greene et al., 1987; Harchol-Balter, Leighton et al., 1999]. Consequently each peer has a set of bloom filters, one for each peer in the system.

A peer can then query for content by querying against these Bloom filters it has received from other peers [Cuenca-Acuna and Nguyen, 2002]. PlanetP uses a

modified vector space ranking model to identify documents that match the query. In a vector space ranking model, documents and queries are represented as vectors. The terms in a document form the different dimensions of the vector and their corresponding weights represent their importance in the document. A comparison of the similarity between a query vector and a document vector can then be used to rank documents that match the query. PlanetP, in addition, also ranks peers in the order of their likelihood to possess relevant documents using an inverse peer frequency measure. This measure ranks peers on the basis of the uniqueness of the terms contained in the index of each peer. Thus, given a query, an ordered list of peers is created and depending on the maximum number of documents specified by the user, relevant documents are returned to the user.

The principle advantage of PlanetP is that it provides a flexible content search mechanism. Though PlanetP is better than schemes that store uncondensed indices, it still suffers from unnecessary storage overhead. This is mainly because each peer stores summaries of the indices of all peers in the system. Therefore, though experiments with PlanetP have indicated that it scales well for systems up to 1000 users, it may not scale well for a much larger system. A possible solution to address this problem could be to enable a peer to choose to combine summaries of peers into a big index, but any query that matches that index will then have to be routed to all the peers in the group resulting in unnecessary traffic. Bandwidth costs are particularly high at system start since peers diffuse their indices simultaneously to all the other peers but gradually decreases with time. Storing indices of all peers in the system not only increases the search accuracy, but also results in reduced search costs, increased reliability, and better performance since queries are directly routed towards peers that maintain the needed resources. Resource availability is, however, still threatened by the departure of peers from the system.

4.2.4.3 SHARK

Symmetric Hierarchy Adaptation for Routing of Keywords (SHARK) is based upon structuring the P2P network and the search space into a multidimensional symmetric redundant hierarchy [Mischke and Stiller, 2003]. SHARK structures the resource metadata in a hierarchical fashion with each level of the hierarchy adding finer granularity to the categorization of the resource. Each level may also have multiple dimensions depending on the degree of categorization required. The meta-data description also includes a string expression that further specifies the object within the lowest level category.

SHARK arranges the system topology so that it exactly matches the structure of the query meta-data (see Figure 12). This enables efficient query routing. Each peer is assigned to a group-of-interest (GoI) according to the objects it stores and to its past request behavior. Each GoI denotes a leaf in the search space hierarchy. In order to ensure that all peers have symmetric roles, the hierarchy adds redundancy by making each peer also assume partial responsibility for its parent positions recursively until the root peer. Therefore each peer is virtually replicated on every level of the hierarchy, and there is no need for a single peer to act as the root.

A SHARK query contains a meta-data description of the desired object (Mq) and thresholds for the minimum required similarity of object and query description for the

structured (tstruct) and the string expression (trand) part of the meta-data description respectively. When a peer receives a query, it looks up its routing table and forwards it to all neighbors whose position meta-data exhibits a similarity with Mq greater than the threshold tstruct. It also adds information on the current level and dimension in the hierarchy that has been resolved to avoid duplication of effort. The query is forwarded until there are no suitable categories or it reaches the corresponding leaf position in the hierarchy. At this point, it is flooded throughout the GoI. Any peer that has a link to a resource with similarity greater than 'trand' returns that link back to the query originator.

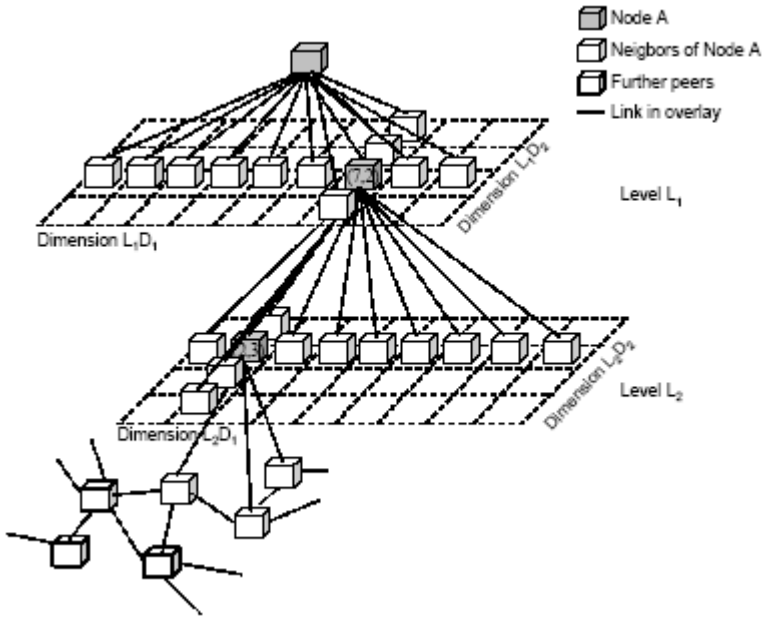


Figure 12: Multidimensional Symmetric Redundant Hierarchy in SHARK

Resource insertion is as follows in SHARK. In order to make a resource available, an insert request with a meta-data description of the resource (MIR) and a replication parameter (r) is routed through the hierarchy. When the query reaches the corresponding leaf GoI, the contacted peer stores a link to the object. If the value of the replication parameter r is greater than 0, a modified query with replication parameter 0 is forwarded to r neighbors.

SHARK supports multi-dimensional search and range searches by using meta-data to describe resources and queries. It provides fault-tolerance and load balancing by enforcing peers to have symmetric roles. And though SHARK is primarily focused around P2P file-sharing applications, it can be used for other applications that involve object search and discovery. Each peer is responsible for storing and maintaining the resources it owns. Categorization of resources allows a query to be forwarded towards a group of peers corresponding to that category. Using meta-data and categories to intelligently route queries helps reduce network traffic and search costs significantly. It also helps better performance and increase search accuracy. Since no peer acts as the single root in the hierarchy, SHARK avoids a single-point-of-failure. This improves

its scalability and reliability. Routing tables maintained by each peer are quite small; consequently, storage cost incurred is quite minimal. These factors contribute towards making SHARK a reliable and scalable search mechanism. SHARK also provides some form of limited redundancy in routing tables to account for the departure of peers. However, since resources are not replicated across multiple peers, resource availability is a bigger problem when peers depart from the system.

Table 3: Comparison of P2P Search Technologies

Technologies / Properties	Blind Search in Unstructured Networks		Informed Search in Unstructured Networks					Search in Structured Networks			Other Search Mechanisms		
	Gnutella	Iterative Deepening	FreeNet	DBFS	Local Indices	Gia	PSP	pSearch	Chord	Gridella	Mobile Agent	PlanetP	SHARK
Decentralized	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes
Local Control	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes
Search Accuracy	*****	*****	******	******	******	*****	******	******	******	******	*****	******	******
Search Flexibility	*	*	*	*	***	***	*	***	*	*	***	****	*****
Performance	*	**	***	***	****	***	****	******	*****	*****	*	*****	*****
Search Cost	******	*****	***	***	***	**	***	*	*	***	*****	*	*
Bandwidth Cost	******	*****	***	***	***	**	**	*	*	***	*	*	*
Storage Cost	*	*	*****	***	***	***	***	*****	****	*****	*****	*****	***
Fault-Tolerance	*	*	***	*	*	***	*	***	*****	***	*	*	***
Scalability	**	****	****	****	****	*****	****	****	******	*****	****	*	*****
Reliability	*	*	***	*	*	***	*	***	*****	***	*	**	***

5 Conclusions

This report surveyed two essential aspects of P2P systems, trust management systems and resource discovery systems. It identified important properties of these two aspects and compared several P2P models and technologies on the basis of these properties. In this section, we present some open research questions that have either not yet been addressed or only partially addressed by the research community. We also describe the contributions of both the trust management and resource discovery aspects and briefly discuss the limitations of this survey.

5.1 Trust Models

We classified trust management systems into three categories: credential-based, reputation-based, and social network-based. Credential-based trust management systems are based on the assumption that service providers and their services can be fully trusted while resource requestors cannot be trusted. Therefore, service providers use credentials and policies to determine the trustworthiness of service requestors. Based on this trust determination, resource requestors acquire access to resources. The focus of reputation-based trust management systems, on the other hand, is on determining the trustworthiness of resource providers either through direct interactions or through information received from other participants of the system. In such systems, it is the resource requestors who are responsible for acquiring information about the reputations of resource providers and their resources before actually accessing them. The third type of trust management systems are based on social networks and rely upon existing social relations among entities and the various roles played by them in order to determine an entity's reputation. Due to its nature of using social relationships, a social network-based trust system can only be applied to certain specific communities.

Our study of different trust management and reputation mechanisms revealed the interplay between the degree of autonomy and the requirement of trust. In a pure centralized system, all control is embedded in a single authority which alone is responsible for making decisions that govern the behavior of all other entities in the system. No other entity makes local decisions; instead, all entities follows the commands of the central authority. As a result, there is an absolute certainty about the actions of each entity and so there is no need to establish trust relationships to determine the trust worthiness of entities. Moving towards a system that promotes increased autonomy to peers leads to a scenario where peers may have separate, possibly conflicting goals and make local autonomous decisions. In such a completely decentralized scenario, there is increased uncertainty about the behavior of peers, and as a result, the confidence that a peer has in other peers is reduced. Trust relationships, therefore, need to be established to determine the trust worthiness of other peers in order to facilitate interaction among them. Autonomy of interacting peers thus results in a need for trust management.

Additionally, we observed that the capabilities of a trust model must increase with the degree of autonomy. There are two main reasons for this. The first reason is that as a system provides greater autonomy to the peers, additional trust mechanisms are needed to determine the trustworthiness of peers. For example, agent-based applications which guarantee certain agent behavior require simpler trust models than online communities

where the behavior of peers is more unpredictable. The second reason is that a trust model requires additional mechanisms to shield peers from the attacks of malicious peers. For example, in open decentralized systems, such malicious peers may try subvert the trust management mechanism itself. The typical approach to this problem has been to facilitate the sharing of trust data among the peers. This may, however, fail because peers are autonomous and may refuse to share invaluable trust data, thus inducing the need for trust models with increased capabilities such as those that support incentive mechanisms [Jurca and Faltings, 2003].

Our survey also identified a taxonomy for trust models and revealed the presence of an ever-increasing body of knowledge about decentralized trust management. We found that the focus of the research community has been either only on policy-based access control systems or on reputation-based systems. Consequently, trust models are only geared towards one of these categories. We believe that trust models belonging to a specific category do not suffice by themselves to provide a complete decentralized trust management solution. Rather, a combination of credential-based and reputation-based trust models enjoys potential benefits and can serve as an excellent starting point for exploring trust management for decentralized systems in the future.

Using the threats of decentralization to survey various trust models revealed to us the existing shortcomings of trust models in the research literature. We believe research in decentralized trust and reputation management is still in its infancy and warrants both an extensive as well as a concentrated research effort in order to be able to completely address these threats. An anticipated benefit of this survey, additionally, is that it will help us identify the essential properties of an ideal decentralized trust model. This is an initial but significant step towards the realization of a consummate reliable decentralized trust model in the future.

Another aspect common to all trust models identified by our survey is the absence of techniques to maintain the anonymity of peers. As described earlier in section 3.1.4, anonymity and trust have an inherent trade-off. However, the research community realizes that protecting the anonymity of peers should be an essential capability of any trust model especially in the case of open decentralized systems. As a result, initial studies have been conducted to explore the trade-off between anonymity and trust [Seigneur and Jensen, 2004]. Several approaches to achieve a balance between trust and anonymity have also been proposed [Dingledine, Mathewson et al., 2003; Bussard, Molva et al., 2004; Pavlov, Rosenschein et al., 2004]. While these reports appear promising, we believe that a substantial work is further needed in this area.

Two other research issues we identified as a result of the survey are the interoperability of trust models, and the relationship between trust models and routing mechanisms which we discuss in the next section. We found several trust models use similar concepts such as conditional transitivity, recommendations or referrals, context-based trust etc. We believe that the identification of these common concepts is a starting step towards achieving trust models interoperability. Attaining this interoperability will enable peers with different trust models to interact in a seamless fashion with each other and provide greater application flexibility.

Another interesting aspect that we discovered is that while there exists an abundance of trust models in the research literature, there is no well-defined way of composing them

modularly into a decentralized application. Some initial approaches have been identified such as the framework proposed by [Gray, O'Connell et al., 2002], but suffer from the shortcomings of having a centralized architecture. Similarly, the effects of using a trust model on the rest of the application have not been extensively investigated. We believe this is essential since trust management is only one aspect of decentralization and so the requirements of a particular application may influence the choice of trust model.

There are a few limitations of our survey. The set of decentralization threats described in this survey is not exhaustive and needs to include several other threats [Damiani, di Vimercati et al., 2002; Lee, Sherwood et al., 2003] in the future in order to provide a comprehensive comparison of trust models. Further, the set of properties used to compare trust models are chosen to elicit only those characteristics of trust models that are of relevant interest to us. Similarly, the set of trust models surveyed in this report are not exhaustive. Trust models discussed are chosen to serve as examples for the taxonomy outlined. In fact, since trust is a cross-cutting concern, several computing communities have focused on the problem of decentralized trust management, albeit in their own respective domains such as multi-agent systems and ubiquitous and pervasive computing [English, Nixon et al., 2002; Shankar and Arbaugh, 2002]. While some of the trust models surveyed in this report belong to different domains (for example, multi-agent systems), there are domains such as ubiquitous computing, grid computing, and mobile *ad hoc* networks that are not covered by this survey.

5.2 Resource Discovery

We classified resource discovery mechanisms in P2P systems as either based on structured networks or unstructured networks. Structured networks are those which make assumptions about the availability of peers and the network, and charge peers with the responsibility of maintaining that structure. They ensure an efficient distribution of data across the peers and guarantee fast and reliable retrieval by efficiently routing queries. Unstructured networks, on the other hand, do not make any assumption about the availability of the peers and treat the arrival and departure of peers from the system as a normal behavior. Peers in unstructured networks store and manage their own data and use comparatively inefficient routing mechanisms to route queries.

Though the taxonomy identified in this survey is not unique as mentioned earlier, it helped us develop insights into the different needs of applications and the technologies best-suited for them. For example, we observed that though models such as Chord and P-Grid afford greater scalability, they are not truly decentralized since peers have to yield local control over the data owned by them. In other words, the system sacrifices the autonomy of peers in order to achieve a common system goal, in this case faster efficient discovery. Clearly, Chord and P-Grid can be used for only those applications where the autonomy of peers is not a necessary constraint. Similarly, systems such as Gnutella and Gia are best suited for applications where peers are autonomous and may not necessarily work together for the common good. Further, in the case of applications that rely on an unstructured network, we observed that applications that use intelligent routing provide faster searches, reduced network traffic, and greater scalability than those that broadcast queries to all the peers, but at the cost of increased storage costs. We concluded, therefore, that the choice of a discovery mechanism depends upon the constraints and requirements of the application.

Drawing upon the technologies surveyed, we classified the efforts of the P2P community in the area of resource discovery into three categories: discovering new technologies or enhancing the various abilities of existing technologies, applying these technologies to new applications, and building tools to support the use of these technologies. It should be noted that these research categories are generic and are applicable to other disciplines of computer science as well. We believe that each of these categories is a fertile ground for extensive research in the domain of P2P resource discovery, but an indepth exploration of issues belonging to these categories is beyond the scope of this paper. Instead we focus below on some of the issues that are relevant to our research.

A main topic of interest to us, also discussed by other researchers [Milojicic, Kalogeraki et al., 2002], is the interoperability of these different discovery mechanisms. While there has been some initial work in this regard [Lui and Kwok, 2002], we believe that this is an area with a lot of substantial benefits. In particular, interoperability will provide peers with the flexibility to use their specific protocols and search mechanisms to interact with other peers. This will enable peers to join any P2P network without having to bother about whether it can interact with other peers in the network.

Another core issue is the composition of these discovery mechanisms into a decentralized application. This is similar to the problem of composing trust models in a decentralized system which we discussed above. While there are several discovery mechanisms utilizing different communication mechanisms, there is no well-defined way of integrating these into a decentralized application. Our current work focuses on architecture-based solutions for this purpose. We believe that an architectural approach will not only help us incorporate these mechanisms but also help us explore how their various abilities can be enhanced. Further, considering the possible correlation between trust models and discovery technologies as described below, an architectural approach will provide guidance on where to incorporate each of these technologies and enable an examination of the relationship between the two.

Finally, our exploration of both trust models and discovery mechanisms in this survey revealed that there exists a definite relationship between the trust model and the choice of a routing scheme. This is because most decentralized trust models depend upon the exchange of trust information between peers. This interaction could be enabled either through broadcast, or multicast, or even point-to-point communication. This motivates three research questions: whether we need exclusive routing schemes for trust data, what are the optimal routing mechanisms to transfer trust data, and whether a peer can be trusted to route trust data. The latter question has also been the focus of Moretown and Twigg [Moreton and Twigg, 2003] who explore the use of trusted routing schemes to increase the robustness of trust management solutions.

A limitation of our survey is that the resource discovery and retrieval technologies discussed are not comprehensive. We have chosen these specific approaches since they serve as representatives of the categories created by the taxonomy. Further the properties used to compare these various technologies are chosen so as to expose only those characteristics that we are interested in. Therefore the comparison of technologies in this survey should not be considered exhaustive.

Figure 13 summarizes the future research concerns for trust management and resource discovery that we have identified in the sections above. As discussed earlier, we have

identified two research issues that cut across both these aspects. The first is that we need to explore the interplay between the routing infrastructure and the choice of a trust model. The second is that we need a well-defined way to integrate these models together in order to construct a decentralized application. The latter issue is the focus of our current research and our initial work has been in identifying the essential elements of decentralization and creating an architectural style to facilitate the incorporation of trust models [Suryanarayana, Erenkrantz et al., 2003] into a decentralized application.

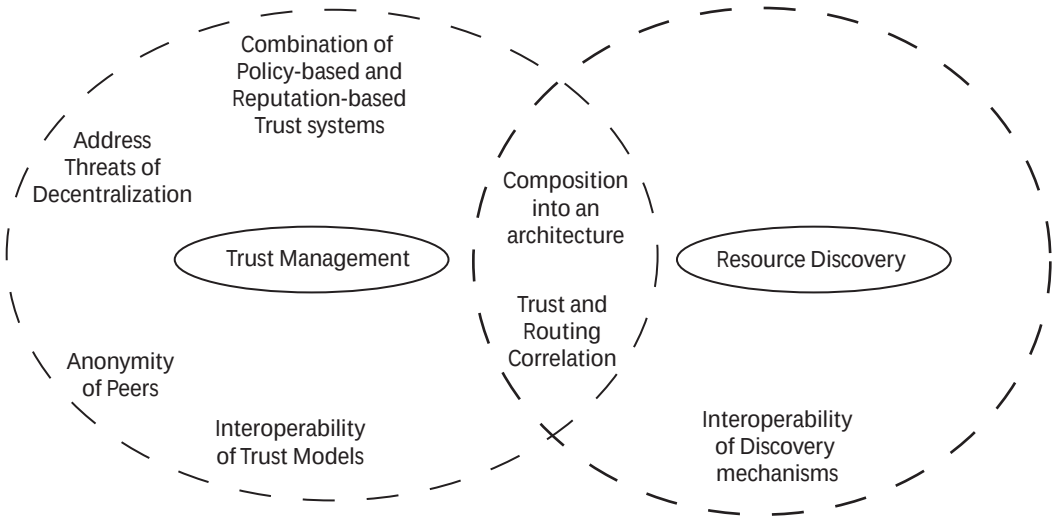


Figure 13: Future Concerns for Trust Management and Resource Discovery

6 Bibliography

[Abdul-Rahman and Hailes, 1997]Abdul-Rahman, A. and Hailes, S. (1997). A Distributed Trust Model. New Security Paradigms Workshop, Langdale, Cumbria UK.

[Abdul-Rahman and Hailes, 2000]Abdul-Rahman, A. and Hailes, S. (2000). Supporting trust in virtual communities. Hawaii International Conference on System Sciences, Maui, Hawaii.

[Aberer, 2001]Aberer, K. (2001). P-Grid: A self-organizing access structure for P2P information systems. 9th International Conference on Cooperative Information Systems, Trento, Italy.

[Aberer and Despotovic, 2001]Aberer, K. and Despotovic, Z. (2001). Managing Trust in a Peer-2-Peer Information System. Conference on Information and Knowledge Management, Atlanta, Georgia.

[Aberer, Punceva et al., 2002]Aberer, K., Punceva, M., et al. (2002). "Improving data access in P2P systems." IEEE Internet Computing Journal: 58-67.

[Bawa, Bayardo Jr. et al., 2003]Bawa, M., Bayardo Jr., R., et al. (2003). Make it fresh, make it quick - Searching a network of personal webservers. World Wide Web, Budapest, Hungary.

[Blaze and Feigenbaum, 1997]Blaze, M. and Feigenbaum, J. (1997). "Managing Trust in an Information Labeling System." European Transactions on Telecommunications **8**: 491-501.

[Blaze, Feigenbaum et al., 1999a]Blaze, M., Feigenbaum, J., et al. (1999a). RFC 2704 - The KeyNote trust-management system version 2.

[Blaze, Feigenbaum et al., 1999b]Blaze, M., Feigenbaum, J., et al. (1999b). The Role of Trust Management in Distributed Systems Security. Secure Internet Programming.

[Blaze, Feigenbaum et al., 1996]Blaze, M., Feigenbaum, J., et al. (1996). Decentralized Trust Management. IEEE Symposium on Security and Privacy.

[Bloom, 1970]Bloom, B. H. (1970). "Space/time trade-offs in hash coding with allowable errors." Communications of the ACM **13**(7): 422-426.

[Bussard, Molva et al., 2004]Bussard, L., Molva, R., et al. (2004). Histroy-Based Signature or How to Trust Anonymous Documents. Second International Conference on Trust Management, Oxford, UK.

[Carzaniga, Rosenblum et al., 2001]Carzaniga, A., Rosenblum, D. S., et al. (2001). "Design and Evaluation of a Wide-Area Event Notification Service." ACM Transactions on Computer Systems **9**(3): 332-383.

[Chawathe, Ratnasamy et al., 2003]Chawathe, Y., Ratnasamy, S., et al. (2003). Making gnutella-like P2P systems scalable. SIGCOMM, Karlsruhe, Germany.

[Chen and Yeager]Chen, R. and Yeager, W. Poblano: A Distributed Trust Model for Peer-to-Peer Networks.

[Chu, Feigenbaum et al., 1997]Chu, Y., Feigenbaum, J., et al. (1997). "REFEREE: Trust management for web applications." World Wide Web Journal: 127-139.

[Clarke, Sandberg et al., 2000]Clarke, I., Sandberg, O., et al. (2000). Freenet: A distributed anonymous information storage and retrieval system. ICSI Workshop on Design Issues in Anonymity and Unobservability, Berkeley, California, USA.

[Cohen and Shenker, 2002]Cohen, E. and Shenker, S. (2002). Replication strategies in unstructured peer-to-peer networks. SIGCOMM, Pittsburgh, Pennsylvania, USA.

[Cuenca-Acuna and Nguyen, 2002]Cuenca-Acuna, F. and Nguyen, T. (2002). "Text-based content search and retrieval in ad hoc P2P communities." Tech. Report DCS-TR-483, Department of Computer Science, Rutgers University.

[Cuenca-Acuna, Peery et al., 2001]Cuenca-Acuna, F., Peery, C., et al. (2001). "PlanetP: Infrastructure support for P2P information sharing." Tech. Report DCS-TR-465, Department of Computer Science, Rutgers University.

[Damiani, di Vimercati et al., 2002]Damiani, E., di Vimercati, S. D. C., et al. (2002). A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. 9th ACM Conference on Computer and Communications Security, Washington DC.

[Deerwester, Dumais et al., 1990]Deerwester, S., Dumais, S., et al. (1990). "Indexing by latent semantic analysis." Journal of the American Society of information Science **41**(6): 391-407.

[Demers, Greene et al., 1987]Demers, A., Greene, D., et al. (1987). Epidemic algorithms for replicated database maintenance. Sixth Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada.

[Deutsch, 1962]Deutsch, M. (1962). Cooperation and Trust: Some Theoretical Notes. Nebraska Symposium on Motivation. M. R. Jones, Nebraska University Press.

[Deutsch, 1973]Deutsch, M. (1973). The Resolution of Conflict: Constructive and Destructive Processes. New Haven, Yale University Press.

[Dingledine, Mathewson et al., 2003]Dingledine, R., Mathewson, N., et al. (2003). Reputation in P2P Anonymity Systems. Workshop on Economics of P2P Systems, Berkeley, CA.

[Dragovic, Hand et al., 2003]Dragovic, B., Hand, S., et al. (2003). Managing trust and reputation in the XenoServer Open Platform. First International Conference on Trust Management, Crete, Greece.

[Dragovic, Kotsovinos et al., 2003]Dragovic, B., Kotsovinos, E., et al. (2003). XenoTrust: Event-based distributed trust management. Second International Workshop on Trust and Privacy in Digital Business, Prague, Czech Republic.

[Dunne, 2001]Dunne, C. (2001). "Using mobile agents for network resource discovery in peer-to-peer networks." ACM SiGecom Exchanges **2**(3).

[English, Nixon et al., 2002]English, C., Nixon, P., et al. (2002). Dynamic Trust Models for Ubiquitous Computing Environments. Workshop on Security in Ubiquitous Computing (UBICOMP 2000), Goteborg, Sweden.

[Foner, 1997]Foner, L. (1997). Yenta: A Multi-Agent, Referral-Based Matchmaking System. First International Conference on Autonomous Agents (Agents '97), Marina del Rey, California.

[Freedman and Morris, 2002]Freedman, M. and Morris, R. (2002). Tarzan: A Peer-to-Peer Anonymizing Network Layer. 9th ACM Conference on Computer and Communications Security, Washington, DC.

[Gambetta, 1990]Gambetta, D. (1990). Trust. Oxford, Blackwell.

[Ghezzi and Vigna, 1997]Ghezzi, C. and Vigna, G. (1997). Mobile code paradigms and technologies: A case study. First International Workshop on Mobile Agents, Berlin, Germany.

[Gnutella]Gnutella gnutella.com.

[Grandison and Sloman, 2000]Grandison, T. and Sloman, M. (2000). "A Survey Of Trust in Internet Applications." IEEE Communications Surveys **3**(4).

[Gray, O'Connell et al., 2002]Gray, E., O'Connell, P., et al. (2002). Towards a Framework for Assessing Trust-Based Admission Control in Collaborative Ad Hoc Applications. Dublin, Ireland, Distributed Systems Group, Department of Computer Science, Trinity College.

[Gupta, Birman et al., 2003]Gupta, I., Birman, K., et al. (2003). Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead. Second International Workshop on Peer-to-Peer Systems, Berkeley, California, USA.

[Gupta, Judge et al., 2003]Gupta, M., Judge, P., et al. (2003). A Reputation System for Peer-to-Peer Networks. Thirteenth ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video, Monterey, California.

[Harchol-Balter, Leighton et al., 1999]Harchol-Balter, M., Leighton, T., et al. (1999). Resource discovery in distributed networks. Eighteenth Annual ACM Symposium on Principles of Distributed Computing, Atlanta, Georgia, USA.

[Hong, 2001]Hong, T. (2001). Performance. Peer-to-Peer: Harnessing the Power of Disruptive Technologies. A. Oram, O'Reilly**:** 203-241.

[Josang and Ismail, 2002]Josang, A. and Ismail, R. (2002). The Beta Reputation System. 15th Bled Electronic Commerce Conference, Bled, Slovenia.

[Jurca and Faltings, 2003]Jurca, R. and Faltings, B. (2003). An Incentive Compatible Reputation Mechanism. IEEE International Conference on E-Commerce, Newport Beach, California, USA.

[Kaashoek and Karger, 2003]Kaashoek, F. and Karger, D. (2003). Koorde: A Simple Degree-Optimal Hash Table. Second International Workshop on Peer-to-Peer Systems, Berkeley, California, USA.

[Kagal, Cost et al., 2001]Kagal, L., Cost, S., et al. (2001). A framework for distributed trust management. Second Workshop on Norms and Institutions in MAS, Autonomous Agents.

[Kamvar, Schlosser et al., 2003]Kamvar, S., Schlosser, M., et al. (2003). The EigenTrust Algorithm for Reputation Management in P2P Networks. WWW, Budapest, Hungary.

[Kan, 2001]Kan, G. (2001). Gnutella. Peer-to-Peer: Harnessing the Power of Disruptive Technologies. A. Oram, O'Reilly**:** 94-122.

[Karger, Lehman et al., 1997]Karger, D. R., Lehman, E., et al. (1997). Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. ACM Symposium on Theory of Computing, ACM Press.

[Khare, 1997]Khare, R., Ed. (1997). Weaving a Web of Trust. World Wide Web Journal, O'Reilly & Associates.

[Khare, 2003]Khare, R. (2003). Extending the REpresentational State Transfer Architectural Style for Decentralized Systems. Information and Computer Science, University of California, Irvine**:** 287.

[Kubiatowicz, Bindel et al., 2000]Kubiatowicz, J., Bindel, D., et al. (2000). OceanStore: An architecture for global-scale persistent storage. Ninth International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, Massachussetts, USA.

[Lacy, Synder et al., 1997]Lacy, J., Synder, J., et al. (1997). Music on the Internet and the Intellectual Property Protection Problem. International Symposium on Industrial Electronics.

[Langley, 2001]Langley, A. (2001). Freenet. Peer-to-Peer: Harnessing the Power of Disruptive Technologies. A. Oram, O'Reilly**:** 123-132.

[Lee, Sherwood et al., 2003]Lee, S., Sherwood, R., et al. (2003). Cooperative peer groups in NICE. IEEE Infocom, San Francisco, USA.

[Li, Mitchell et al., 2002]Li, N., Mitchell, J., et al. (2002). Design of a role-based trust management framework. IEEE Symposium on Security and Privacy, Oakland, California.

[Luhmann, 1979]Luhmann, N. (1979). Trust and Power. New York, John Wiley.

[Lui and Kwok, 2002]Lui, S. and Kwok, S. (2002). "Interoperability of Peer-to-Peer File Sharing Protocols." ACM SiGecom Exchanges **3**(3): 25-33.

[Lv, Cao et al., 2002]Lv, Q., Cao, P., et al. (2002). Search and replication in unstructured peer-to-peer networks. International Conference on Supercomputing, New York, USA.

[Marsh, 1994]Marsh, S. (1994). Formalising Trust as a Computational Concept. Department of Mathematics and Computer Science. Stirling, University of Stirling.

[Maymounkov and Mazieres, 2002]Maymounkov, P. and Mazieres, D. (2002). Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. First International Workshop on Peer-to-Peer Systems, Cambridge, Massachussets, USA.

[Menasce and Kanchanapalli, 2002]Menasce, D. and Kanchanapalli, L. (2002). "Probabilistic scalable P2P resource location services." ACM SIGMETRICS Performance Evaluation Review **30**(2).

[Milojicic, Kalogeraki et al., 2002]Milojicic, D., Kalogeraki, V., et al. (2002). "Peer-to-peer computing." Technical Report HPL-2002-57 Hewlett Packard Laboratories.

[Minsky, 2003]Minsky, N. H. (2003). Regularity-Based Trust in Cyberspace. First International Conference on Trust Management, Crete, Greece.

[Mischke and Stiller, 2003]Mischke, J. and Stiller, B. (2003). Rich and scalable peer-to-peer search with SHARK. Autonomic Computing Workshop Fifth Annual International Workshop on Active Middleware Services, Seattle, Washington, USA.

[Moreton and Twigg, 2003]Moreton, T. and Twigg, A. (2003). Enforcing Collaboration in Peer-to-Peer Routing Services. First International Conference on Trust Management, Crete, Greece.

[Napster]Napster http://www.napster.com.

[Pavlov, Rosenschein et al., 2004]Pavlov, E., Rosenschein, J., et al. (2004). Supporting Privacy in Decentralized Additive Reputation. Second International Conference on Trust Management, Oxford, UK.

[Pujol, Sanguesa et al., 2002]Pujol, J., Sanguesa, R., et al. (2002). Extracting reputation in multi agent systems by means of social network topology. First International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy.

[Ratnasamy, Francis et al., 2001]Ratnasamy, S., Francis, P., et al. (2001). A Scalable Content Addressable Network. Proceedings of ACM SIGCOMM, San Diego, CA.

[Reed, Syverson et al., 1996]Reed, M., Syverson, P., et al. (1996). Proxies For Anonymous Routing. Twelfth Annual Computer Security Applications Conference, San Diego, California.

[Resnick, Zeckhauser et al., 2000]Resnick, P., Zeckhauser, R., et al. (2000). "Reputation Systems." Communications of the ACM **43**(12): 45-48.

[Rhea, Eaton et al., 2003]Rhea, S., Eaton, P., et al. (2003). Pond: The OceanStore prototype. Second USENIX Conference on File and Storage Technologies, San Francisco, California, USA.

[Ritter]Ritter, J. "Why gnutella can't scale? No, really." http://www.tch.org/gnutella.html.

[Rowstron and Druschel, 2001a]Rowstron, A. and Druschel, P. (2001a). Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-Peer Systems. Eighteenth IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), Heidelberg, Germany.

[Rowstron and Druschel, 2001b]Rowstron, A. and Druschel, P. (2001b). Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. Eighteenth ACM Symposium on Operating System Principles, Banff, Alberta, Canada.

[Sabater and Sierra, 2001]Sabater, J. and Sierra, C. (2001). REGRET: A Reputation Model for Gregarious Societies. 4th Workshop on Deception, Fraud and Trust in Agetn Societies, Montreal, Canada.

[Sabater and Sierra, 2002]Sabater, J. and Sierra, C. (2002). Reputation and social network analysis in multi-agent systems. First International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy.

[Scarlata, Levine et al., 2001]Scarlata, V., Levine, B., et al. (2001). Responder anonymity and anonymous peer-to-peer file sharing. IEEE International Conference on Network Protocols, Riverside, California, USA.

[Schillo, Funk et al., 2000]Schillo, M., Funk, P., et al. (2000). "Using trust for detecting deceitful agents in artificial societies." Applied Artificial Intelligence Journal, Special Issue on Trust, Deception and Fraud in Agent Societies.

[Seigneur and Jensen, 2004]Seigneur, J.-M. and Jensen, C. (2004). Trading Privacy for Trust. Second International Conference on Trust Management, Oxford, UK.

[Shankar and Arbaugh, 2002]Shankar, N. and Arbaugh, W. (2002). On Trust for Ubiquitous Computing. UBICOMP2002 - Workshop on Security in Ubiquitous Computing, Goteborg, Sweden.

[Shields and Levine, 2000]Shields, C. and Levine, B. (2000). A Protocol for Anonymous Communication Over the Internet. Seventh ACM Conference on Computer and Communications Security, Athens, Greece.

[Stoica, Morris et al., 2001]Stoica, I., Morris, R., et al. (2001). Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. ACM SIGCOMM, San Diego, California, USA.

[Suel, Mathur et al., 2003]Suel, T., Mathur, C., et al. (2003). ODISSEA: A peer-to-peer architecture for scalable web search and information retrieval. International Workshop on the Web and Databases, San Diego, California, USA.

[Suryanarayana, Erenkrantz et al., 2003]Suryanarayana, G., Erenkrantz, J., et al. (2003). "PACE: An architectural style for trust management in decentralized applications." ISR Technical Report UCI-ISR-03-9, Institute of Software Research, UC Irvine.

[Suryanarayana, Erenkrantz et al., 2004]Suryanarayana, G., Erenkrantz, J. R., et al. (2004). PACE: An Architectural Style for Trust Management in Decentralized Applications. 4th Working IEEE/IFIP Conference on Software Architecture, Oslo, Norway.

[Suryanarayana and Taylor, 2002]Suryanarayana, G. and Taylor, R. N. (2002). A Decentralized Algorithm for Coordinating Independent Peers: An Initial Examination. Tenth International Conference on Cooperative Information Systems (CoopIS), Irvine, California.

[Tang, Xu et al., 2003]Tang, C., Xu, Z., et al. (2003). Peer-to-peer information retrieval using self-organizing semantic overlay networks. SIGCOMM, Karlsruhe, Germany.

[Tang, Xu et al., 2002]Tang, C., Xu, Z., et al. (2002). "PeerSearch: Efficient information retrieval in peer-to-peer networks." Tech. Report HPL-2002-198, Hewlett-Packard Labs.

[Tarjan, 1972]Tarjan, R. E. (1972). "Depth-first search and linear graph algorithms." SIAM Journal on Computing **1**(2): 146-160.

[Tsoumakos and Roussopoulos, 2003]Tsoumakos, D. and Roussopoulos, N. (2003). "Analysis and comparison of P2P search methods." Technical Report CS-TR-4539, UMIACS-TR-2003-107, Department of Computer Science, University of Maryland.

[Yang and Garcia-Molina, 2002]Yang, B. and Garcia-Molina, H. (2002). Improving search in peer-to-peer networks. International Conference on Distributed Computing Systems, Vienna, Austria.

[Yao, 2003]Yao, W. (2003). Fidelis: A Policy-Driven Trust Management Framework. First International Conference on Trust Management, Crete, Greece.

[Yu and Singh, 2000]Yu, B. and Singh, M. P. (2000). A social mechanism of reputation management in electronic communities. Fourth International Workshop on Cooperative Information Agents.

[Yu, Winslett et al., 2001]Yu, T., Winslett, M., et al. (2001). Interoperable strategies in automated trust negotiation. 8th ACM Conference on Computer and Communications Security, Philadelphia, USA.

[Zacharia and Maes, 1999]Zacharia, G. and Maes, P. (1999). Collaborative Reputation Mechanisms in Electronic Marketplaces. 32nd Hawaii International Conference on System Sciences, Hawaii.

[Zacharia and Maes, 2000]Zacharia, G. and Maes, P. (2000). "Trust Management Through Reputation Mechanisms." Applied Artificial Intelligence **14**: 881-907.

[Zhao, Huang et al., 2003]Zhao, B., Huang, L., et al. (2003). "Tapestry: A resilient global-scale overlay for service deployment." IEEE Journal on Selected Areas in Communication.

[Zhao, Kubiatowicz et al., 2001]Zhao, B., Kubiatowicz, J., et al. (2001). "Tapestry: An infrastructure for fault-tolerant wide-area location and routing." Tech. Report UCB/CSD-01-1141.

[Zhuang, Zhao et al., 2001]Zhuang, S., Zhao, B., et al. (2001). Bayeux: An architecture for scalable and fault tolerant wide-area data dissemination. Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video, Port Jefferson, New York, USA.

[Zimmermann, 1994]Zimmermann, P. (1994). PGP User's Guide, MIT Press, Cambridge.