A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks

Ernesto Damiani
DTI - Università di Milano
26013 Crema - Italy
damiani@dti.unimi.it

De Capitani di Vimercati
DEA - Università di Brescia
25123 Brescia - Italy
decapita@ing.unibs.it

Stefano Paraboschi
DEI - Politecnico di Milano
20133 Milano - Italy
parabosc@elet.polimi.it

Pierangela Samarati
DTI - Università di Milano
26013 Crema - Italy
samarati@dti.unimi.it

Fabio Violante
DEI - Politecnico di Milano
20133 Milano - Italy
violante@elet.polimi.it

ABSTRACT

Peer-to-peer (P2P) applications have seen an enormous success, and recently introduced P2P services have reached tens of millions of users. A feature that significantly contributes to the success of many P2P applications is user anonymity. However, anonymity opens the door to possible misuses and abuses, exploiting the P2P network as a way to spread tampered with resources, including Trojan Horses, viruses, and spam. To address this problem we propose a self-regulating system where the P2P network is used to implement a robust reputation mechanism. Reputation sharing is realized through a distributed polling algorithm by which resource requestors can assess the reliability of a resource offered by a participant before initiating the download. This way, spreading of malicious contents will be reduced and eventually blocked. Our approach can be straightforwardly piggybacked on existing P2P protocols and requires modest modifications to current implementations.

Categories and Subject Descriptors

C.2.0 [**Computers-Communication Networks**]: General—*Security and protection*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Data sharing*; K.6.5 [**Computers and Society**]: Security and Protection—*Invasive software*

General Terms

Security, Design

Keywords

Peer-to-peer network, reputation-based systems, polling protocol

CCS'02, November 18-22, 2002, Washington, DC, USA.

1. INTRODUCTION

Peer-to-peer (P2P) is a general label that was originally used to identify network protocols where all the nodes have the same role and there are no nodes with a special responsibility to monitor or supervise the network behavior. Recently, the term has been mainly used to identify a family of applications that exploit the Internet to offer services where each participant acts both as a client and as a resource provider. Current P2P applications can be classified into one of the following three categories.

1. *File sharing*: P2P applications such as Napster, Gnutella, FreeNet, KaZaA and others that make it possible for Internet users to share files.

2. *Distributed processing*: rather than sharing files, P2P systems for distributed processing share the computational power of their nodes; a well-known example is SETI@Home.

3. *Instant messaging*: client programs such as MSN Messenger, Yahoo! Messenger, and AOL Instant Messenger (AIM) allow users to exchange text, voice messages, and files.

The focus of our paper will be on P2P applications for file exchange. Several worries have been raised about the use of P2P file sharing applications. A typical concern is performance-related: the huge audio and video files that users typically share could clog the global Internet as well as corporate networks. These issues can be kept under control with relative ease by *traffic-shaping* techniques that enforce bandwidth constraints on P2P traffic. A greater and more difficult problem of P2P file sharing applications is that they introduce a whole new class of security threats, as they can be exploited to distribute malicious software, such as viruses and Trojan horses, even bypassing the protections of firewalled networks. This risk is not only present when a user downloads executable content. Indeed, also audio and video files may harbor security threats, as the multimedia formats permit the introduction of links and active content that may be exploited to introduce malicious software into a computer.[1] Also, there are many applications of P2P technologies where the network is used to distribute executable content. For instance, one of the most successful P2P applications, KaZaA desktop [16], is distributed by the P2P network realized by the application itself. Antivirus software producers are considering P2P networks as a convenient way to distribute virus signature updates, exploiting the resiliency and aggregate bandwidth of P2P networks, and avoiding the overload of central Web servers. On the other hand, P2P content distribution involves spamming, which applies to any possible content; spammers may find P2P networks a convenient way to distribute their unrequested promotional material, offering fake resources able to elicit the interest of users.

These P2P security challenges cannot be fully met by means of traditional techniques. For instance, prohibitions on the use of P2P applications, by organization policy or blocking ports used by file sharing P2P programs, may not be effective. As a matter of fact, users often show a particular interest in P2P applications and may use them anyway, configuring their programs for the use of unblocked ports. A technical solution that permits users to responsibly choose the level of risk of their actions appears to us more appropriate, as it has greater chance to be supported by users and allows to exploit the benefits of P2P technologies.

In the typical Web environment, users estimate the level of risk in a resource using the perceived reputation of its source. Users may choose to trust only resources offered by servers with a good reputation, and they will avoid resources offered by servers that have a bad reputation. When they have no way to attribute a good or bad reputation to a server, the choice depends on the level of interest in the resource and on the risk that they are ready to take. What is natural for the Web, becomes a difficult problem for an anonymous P2P environment, where resource providers are identified by a pseudonym and an IP address.

Previous proposals supporting the concept of reputation in P2P scenarios (e.g., [2, 8, 10]) associate reputations with servents. In this paper, we propose an approach that uses combined reputations of servents and resources, providing more informative pollings and overcoming the limitations of servent-based only solutions. Servent reputations are associated with the servent identifier, which has to be tamper resistant. Resource reputations are tightly coupled to the resources' content via their *digest*, thus preventing their forging on the part of malicious peers. Reputations are cooperatively managed via a distributed polling algorithm in order to reflect the community's view of the potential risk involved with the download and use of a resource. We present a protocol, called XRep, for maintaining and exchanging reputations that can be straightforwardly piggybacked on existing P2P protocols, and discuss the advantages it provides against known attacks to P2P networks. We discuss the comparative advantages and limitations of resource-based vs. servent-based reputation solutions and therefore how their combined use can result advantageous. Finally, we presents some experimental results, based on the Gnutella framework, to analyze and validate the viability of our approach. It is important to note that, while it can be easily implemented on top of existing systems like Gnutella (as in our current implementation), our technique is virtually independent from the architectural details of the P2P environment and can therefore be applied to different solutions as well.

[1]See, for example, the BugTraq message on April 2002, describing how an MP3 file played by WinAmp could be configured to execute arbitrary commands, `http://online.securityfocus.com/archive/1/269724`.

2. P2P INFORMATION SHARING ARCHITECTURES

P2P file sharing applications' success is testified by the millions of users currently contributing to Internet-based file exchange networks. At the time of writing (September 2002), more than 100 million copies of the KaZaA file sharing application [16] have been downloaded (currently increasing at the rate of more than 3 million downloads every week). Several billions of file exchanges occur monthly on file sharing networks.

Since nodes in a P2P network play the role of both *ser*ver and cli*ent*, the neologism *servent* has been introduced to identify this double responsibility. A servent in the network acts as a server when offering its resources, answering queries, and possibly dispatching them to neighboring nodes; it acts as a client when the user issues a request and retrieves resources from other servents.

A P2P file exchange involves two phases: *search* and *download*. The search phase is aimed at looking for a servent offering a given file. In the download phase a direct connection is established with the servent offering the searched file and a download is started. The download phase is relatively uniform across different P2P applications; usually, the traditional protocols for file exchange of the TCP/IP family (FTP, HTTP) are used; a notable variant is the parallel download that is realized by many recent applications. What most characterizes each application is the solution adopted for the search phase, which can be implemented in three ways: *pure P2P*, *centralized index*, and *distributed architecture with supernodes*. In this paper, we are not concerned with this issue and the impact that the architectural choice may have on the implementation of our protocol. We refer our work to the pure P2P architecture which is closest to the ideal structure of the peer-to-peer spirit, where all participants have uniform role. In particular, we will refer to the Gnutella architecture [21]. In the Gnutella architecture, each node connects to the network by choosing a number of hosts to which it is directly linked. This set of connections creates a logical file exchange network overlaid on the TCP/IP network.

To look for a file, a servent p sends a broadcast `Query` message to every node to which it is directly linked. Servents identifying the requested file in their repository answer with a `QueryHit` message which is returned to the connection from which the request arrived. Even if the servent identifies the resource in its repository, it will forward the request along all the links it maintains, except the one from which the request arrived. The `QueryHit` message contains the *ResultSet* and the pair ⟨IP address, port⟩ that must be used to download the file via HTTP.

In other words, each Gnutella node operates as a router for queries traveling along the P2P network. To avoid overloading the network, each node configures a *Time-To-Live* (TTL) for messages, which represents the number of nodes that each message can pass through. On passing through a node, the TTL of a forwarded message is decreased by one;

when the TTL reaches zero, the message is dropped. The trace of message identifiers is also used to route `QueryHit` messages back to the node where the corresponding `Query` message originated. The limit on the TTL introduces a limit on the network nodes' reachability, called *horizon.* Each servent will normally have the possibility to interact with only a portion of the nodes of the Gnutella network. A node's horizon depends on the number of connections that the node opens with its neighbors (typical values are in the range 2 to 6) and on the TTL it sets on messages. These values are chosen considering the bandwidth available directly to the node and to the network infrastructure. The choice is a tradeoff between the benefit deriving from a greater number of connections, with an extension of the horizon, and the increase in processing time and bandwidth required to receive, evaluate, and dispatch the queries in transit.

3. XREP PROTOCOL

We now discuss the basic assumptions of our solution, and present our XRep protocol.

3.1 Basic assumptions

Our approach extends Gnutella-like environments by providing facilities for assigning, sharing, and combining reputations on servents and resources. As illustrated in Section 2, in a Gnutella-like system, a servent p looking for a resource broadcasts a `Query` message and receives back a set of replies from which it chooses the resource to download and the servent from which to download. Typically, resource selection is based on the offer quality (e.g., the number of hits and declared connection speed) or on preference criteria based on the requestor's past experiences. Our approach is to allow p to enrich its choice selection process by inquiring the network for peers' opinions (*votes*) on resources and their offerers.

To be able to refer to resources and servents (and to allow for checking correspondence of a servent with a declared identifier) we make the following two assumptions. First, we require the *servent_id* associated with a servent to be a digest of a public key, obtained using a secure hash function [4] and for which the servent knows the corresponding private key. Second, each resource is associated with an identifier (*digest*) computed applying a secure hash function to the resource content [15]. These assumptions do not imply loss of anonymity, as the digest associated with a servent is only a pseudonym (*opaque identifier*). Also, servents are not required to keep their identifiers persistent; persistence of an identifier simply allows the servent to build a reputation for it, which will be nullified if the identifier is changed.

The basic idea of our approach is that each servent maintains information on its own experience on resources and other servents, and can share such experience with others upon request. While many solutions can be taken into account with respect to how to store, maintain, and share such information, in this paper we consider a specific approach (which we adopted in our current implementation) and assume that each peer maintains two *experience repositories*:

- a *resource repository*, which is a table with attributes (*resource_id,value*) associating, with each *resource_id* the peer has experienced, a binary value describing whether the resource is good (+) or bad (−) in the peer's opinion.
- a *servent repository*, which is a table with attributes (*servent_id,num_plus,num_minus*) associating, with each *servent_id* the peer has interacted with, the number of successful and unsuccessful downloads.

The precise semantics of the values in the tables is outside the scope of the paper as our approach is independent from it. As for the resource repository we simply interpret a '+' as a satisfaction of the peer in the resource, and '−' as the lack thereof. Non-satisfaction could reflect the fact that the resource did not completely fulfill the expectations of the peer or even that it was corrupted or malicious. At the minimum, the value associated with a resource should capture the trust of the servent in the resource integrity (e.g., that the resource does not contain a virus). An analogous reasoning can be followed for the positive and negative counters associated with each servent in the servent repository.

The values maintained in the repositories are used to express votes on resources and servents in the framework of XRep. We assume votes to be binary and encoded as `1` and `0`, where `1` expresses a positive opinion on a resource/servent and `0` expresses a negative one.[2] Again, there are different ways for translating recorded experience into votes and it is up to the Gnutella servent configurator to decide the specific choice to adopt. For instance, a servent could vote `1` only on servents for which *num_minus* is zero or on servents for which *num_plus* is much higher than *num_minus*.

It is worth noting that votes need not be binary and that servents need not agree on the scale on which to express them. For instance, votes could be expressed in an ordinal scale (e.g., from `A` to `D` or from $*****$ to $*$) or in a continuous one (e.g., a servent can consider a resource reliable at 80%). The only constraint for our approach to work properly is that the scale on which one expresses votes should be communicated to the poller. Also, while we assume votes to be monodimensional, multidimensional evaluations (e.g., *feature vectors* [20]) could be considered.

3.2 Polling protocol

The XRep protocol, illustrated in Figure 1, consists of the following phases: *resource searching*, *resource selection and vote polling*, *vote evaluation*, *best servent check*, and *resource downloading*.

Phase 1: Resource searching. First, like in a standard Gnutella interchange, the initiator p broadcasts to all its neighbors a `Query` message containing the search keywords. Again as in the standard Gnutella protocol, when a servent receives a `Query` message for which it has a match, it responds with a `QueryHit` message. The `QueryHit` message includes the number of files *num_hits* that matched the keywords, a set *ResultSet* of triples containing the files' names and related information, the *speed* in Kb/second of the responder, the *servent_id* of the responder, and the pair $\langle IP,port \rangle$ to be used to download the files. In addition to this standard information, our enhanced `QueryHit` also reports the digests associated with the resources named in *ResultSet*. To this end, we exploit the auxiliary *trailer* field, an extension first introduced by BearShare v1.3.0 for carrying application-dependent information, and described in the Gnutella Protocol Specification v0.4 [21].

[2] The syntactic distinction between the values in the repositories and in the integer encoding for the votes is meant to express their different semantics.

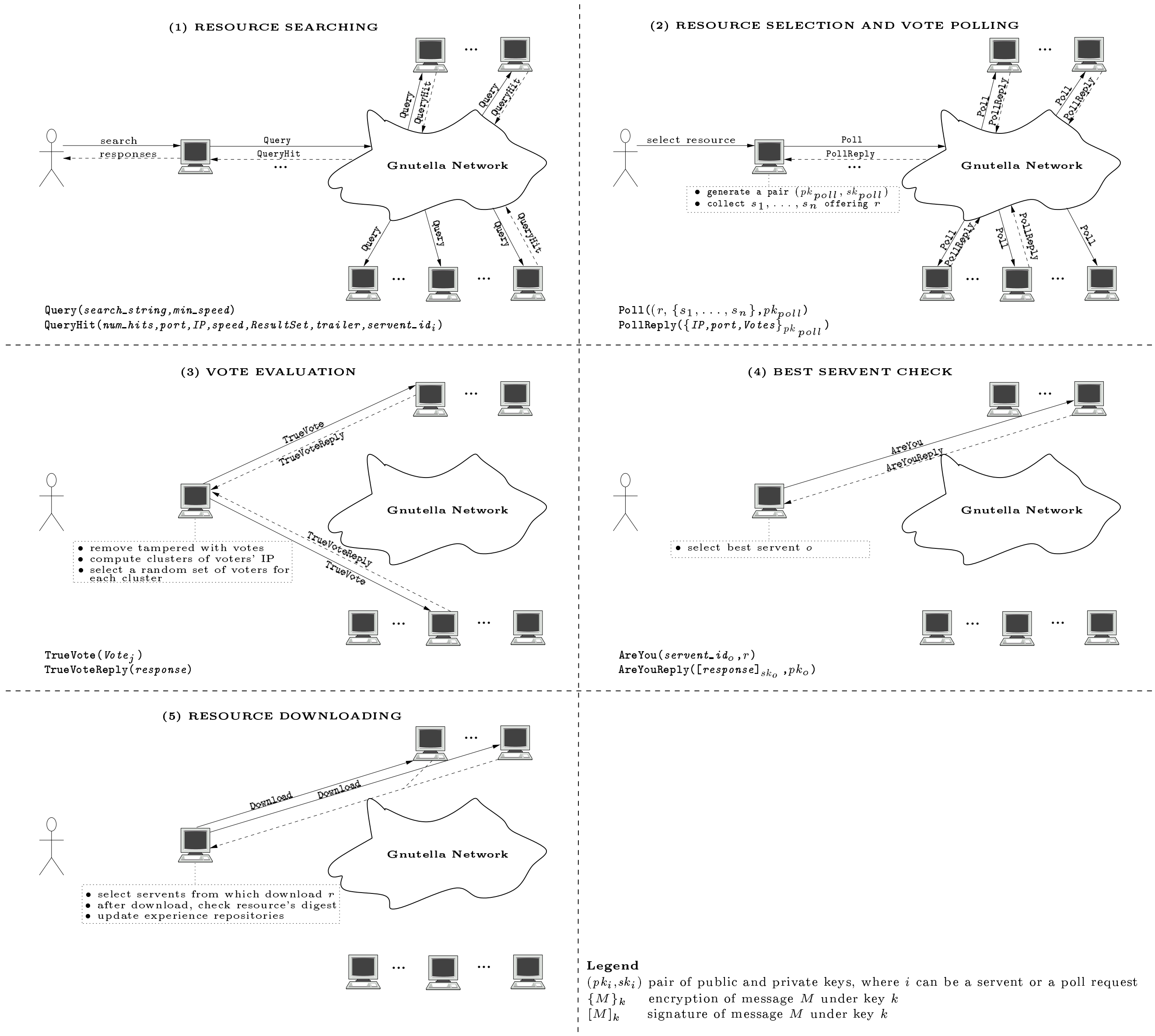

Figure 1: Sequence of messages and operations in the XRep protocol

Phase 2: Resource selection and vote polling. Upon reception of the `QueryHit`s, the initiator p selects, among the possibly different resources offered, the resource r that best seems to satisfy its request. Such selection can be guided by the user's preferences and/or by the number of offerers.

At this point our solution is to allow p to inquire its peers about the downloading it is about to execute. Such polling could be done by asking peers their opinion on either the resource or the servents that offer it. As either one of the choices by itself provides some shortcomings (see Section 5) we combine the two solutions and assume p sends its peers a vote request on resource r as well as on the servents $s_1, \ldots, s_n$ that offer it.

Vote polling is performed via broadcasting on the Gnutella network. `Poll` messages are implemented on top of ordinary `Query` messages. To protect the integrity and confidentiality of poll responses, the poll request also includes a public key $\textsc{pk}_{poll}$, with which poll responses will need to be encrypted. Such a public key can be generated on the fly for each poll request or be a persistent key that p can use multiple times. (Note that our use of public key encryption does not require a central CA or a higher authority, as there is no need to establish and certify the correspondence between a pair of keys and an identity. Our only requirement is that p be the only one to know the corresponding private key.)

Upon receiving a `Poll` message, each peer checks its *experience repositories* and can respond communicating its votes on the resource as well as on the servents. Votes are communicated back to the poller via the Gnutella network as a `PollReply` message exploiting the `QueryHit` message.

The message, encrypted with key $\textsc{pk}_{poll}$, includes the peer's votes, together with its IP and port. As said above, encryption will allow p to detect if the message has been tampered with while in transit; also it preserves the confidentiality of the votes and their association with those who have expressed them.

Phase 3: Vote evaluation. As a result of the previous phase, p collects a set of votes on the targeted resources and its offerers. To base its decision on the votes received, p needs to trust the reliability of the votes and recognize (and possibly collapse) votes that might belong to a clique. This process is composed of three steps. First, p uses decryption to detect tampered with votes and discards them. Second, in order to recognize cliques of dummy or controlled voters (e.g., pseudospoofing attacks, cf. Section 4) p clusters the voters' IPs, for instance collapsing together those sharing the same `net_id`. Cluster computation allows p to properly weight the votes within a cluster (e.g., by considering a single final vote for each cluster, or taking an average weighted on the cluster's size). Third, p randomly selects a set V' of voters within each cluster, and directly contacts each $v_j \in V'$ (using the IP address and port received for it) with a `TrueVote` message[3] requesting confirmation on the votes that p has received from it. Each contacted voter is required to send a `TrueVoteReply` message for vote confirmation. This forces the attacker to pay the cost of using real IPs as false witnesses (e.g., shilling attacks, cf. Section 4). If not enough of the contacted voters respond positively (within a given time frame), the random selection process is repeated.

After this checking process, p can trust its assessment of the resources reputation (or its offerers), and therefore finally decides to download it. If p judges the evidence on the resource quality (or its offerers) not sufficient, it can repeat the voting process on another resource. In principle, the voting process can be executed on a list of top resources and the resource to download can be selected by comparing the votes received.

Phase 4: Best servent check. Once having taken the decision of downloading the resource, p should select the offerer from which to execute the download. While one may think that any offerer would be fine (as they are all offering the same hash) a blind decision could expose p to attacks of servents exploiting someone else's good reputation for offering bad resources (e.g., ID stealth attacks, cf. Section 4). Reliability of servents must then also be taken into account in the choice. A straightforward way to satisfy this criteria is for p to download the resource from the offerer who has the best reputation (according to the votes received). The drawback of such a solution is that it could easily create a performance bottleneck on reliable servents. To avoid this, our protocol assumes a phase where the most reliable servent is contacted to check the fact that it exports the resource. Once this has been assessed, p can rest sure that the resource digest is in fact reliable.

Phase 5: Resource downloading. At this point p decides from which servent to download the resource. Then, it contacts the chosen servent directly and requests the resource. After the download, p will check the resource against its digest to ensure its integrity (discarding the resource if there is no correspondence). Also, p will update its repositories with its opinion on the downloaded resource and its offerer.

It is important to note that the decoupling of the resource from the offerers (provided by Phase 4) permits the execution of parallel downloads, whereby p can ask different offerers for different fragments of the resource to be downloaded.

[3]While in Figure 1 votes are transmitted in clear, an alternative solution considering their encryption with a key communicated by the voter in its `PollReply` message can be considered.

4. SECURITY CONSIDERATIONS ON XREP

P2P networks have been designed to distribute the responsibilities traditionally held by centralized servers across the network, delegating them to the clients. But, as P2P systems disperse resources into the network, they also disperse security weaknesses. Besides malicious attacks by peers actively trying to spread spam or hostile content, information sharing via P2P networks is also vulnerable to "unintentional" attacks by well-meaning participants, redistributing a resource which was tampered with without their knowledge. Tampering is not restricted to executable content; it may affect files that contain multimedia information which is different than their title indicates, low quality, or simply unsolicited spamming. One of the main goals of our XRep protocol is to improve the global security and quality of content distribution within P2P networks. XRep allows us to protect P2P networks against most known attacks as discussed in the following. We distinguish between general weaknesses of P2P systems and weaknesses specifically referred to reputation-based approaches.

4.1 Attacks to P2P systems

The two major general vulnerabilities of content distribution via P2P networks are *self replication* and *man in the middle* attacks.[4]

Self replication. The simplest version of this attack is based on the fact that in current P2P systems there is virtually no way of verifying the source or content of a message. Most P2P systems manage their own namespace, allowing users to have temporary identities on the system, regardless of their IP address. Such identities are usually not persistent and, in principle, could change at every interaction. Of course, every peer has an IP address, and perhaps even a DNS name associated with it; but IP-based identification is hardly feasible when the binding between a peer and its IP address is made via dynamic *Network Address Translation* (NAT).[5]

Under the protection of this relative anonymity, a malicious peer could answer positively to all queries, and then return doctored content. Even honest peers, unaware of the malicious content of the doctored resource, could share it, contributing to its diffusion. For instance, a Gnutella worm called *Mandragore* registers itself as an active peer within the network, responding affirmatively to all intercepted requests. As an answer, it provides a renamed copy of itself for downloading. In our protocol, after a few hits, the Mandragore's digest will start being associated with bad votes

[4]Other weaknesses, related to disclosure of confidential information such as backdoor and firewall-traversal related ones [17], are intrinsic to the usage of the P2P architecture in corporate networks and outside the scope of the paper.

[5]Presumably, this situation will be alleviated with IPv6, but its widespread adoption will take a long time.

and therefore its subsequent downloading prevented, regardless of the peer which is offering it. In other words, being forced to provide fake resources with their own digest implies that the offered resource will have a bad reputation and therefore it is unlikely that it will be downloaded (Phase 3 of Figure 1).

Note also that the typical countermeasure of viruses to digest verification is slightly modifying themselves while preserving their functionality. While this behavior would avoid collecting negative votes on a single digest, it will however not provide the virus with the ability to collect positive recommendations and therefore be selected. Self replication attacks could then only work at cold-start, that is, when sharing resources in response to previously unheard-of keywords. Our approach does explicitly address this setting, collecting votes on both resources and servents rather that on resources or servents alone [8].

Man in the middle. This kind of attacks takes advantage of the fact that, due to application-level routing of P2P networks, a malicious peer can lie in the path between two "honest" peers. The basic version of the attack works as follows. Assume that A is a peer searching for a file, B is a peer that has the file A is looking for, and D is a malicious peer.

- A broadcasts a `Query` message and B responds.
- Malicious peer D intercepts the `QueryHit` message from B and modifies the IP and port fields to contain D's IP address and port. The modified `QueryHit` message is then sent back to A.
- A decides to download the file from D, which provides a fake resource (possibly even a hostile version of the original one provided by B).[6]

Our protocol neutralizes this attack because first of all it is unlikely that D will be selected, and even if so, the fake or doctored content provided by D will not match the digest of the legitimate resource, and will therefore be discarded by the recipient A (Phase 5 of Figure 1).

4.2 Attacks to reputation-based systems

Recent attacks specifically aimed at reducing the effectiveness of reputation-based systems [17] include *pseudospoofing*[7], *ID stealth*, and *shilling*.

Pseudospoofing. The pseudospoofing attack exploits the P2P systems use of pseudonyms. Pseudospoofing attackers create and control multiple phony identities. This technique can be readily used to compromise P2P systems relying on pseudonyms rather than on full authentication. Pseudospoofing is relatively resilient to countermeasures based on servent reputations alone, as a malicious peer can turn to a new pseudonym after earning a bad reputation, and simulate multiple witnesses willing to give fake evidence in its favor. However, abandoning pseudonyms entirely would require a complex and expensive infrastructure capable of verifying the legal identities of all participants. The digest-based mechanism of our approach makes pseudospoofing unprofitable, as doctored resources can be recognized and discarded (Phase 5 of Figure 1). Also, using pseudospoofing to build cliques of fake witnesses is prevented by the IP checking mechanism hardwired in our protocol. If fake identities are simulated as pseudonyms, all of them would fall in a single IP cluster (Phase 3 of Figure 1). If fake IPs are also provided by the attacker, they are likely to be discovered by the `TrueVote`/`TrueVoteReply` message exchange (Phase 3 of Figure 1).

ID Stealth. ID stealth is a variant of the pseudospoofing attack that can be used to reduce the effectiveness of reputation management techniques based on voting protocols, like the one described in this paper. Upon receiving a `Query` message from a requestor A, the attacker answers with two or more `QueryHits`, all of them carrying the digest of a doctored or malicious resource. One of the `QueryHits` carries the attacker's ID (say, C) while the others carry IDs that have been "stolen" by the attacker from reputable servents. Here, for the sake of simplicity, we assume attacker C to use only one stolen ID, called B. After receiving the two `QueryHits` carrying respectively IDs B and C and the same malicious content's digest, A polls the network about the digest and the two candidate servents, collecting several (positive) votes about servent B, while both C and the resource digest turn out to be unknown. Relying on the outcome of the voting procedure, A starts downloading the resource from the only available servent (C) and receives malicious content perfectly matching the digest. Phase 4 of Figure 1 is aimed at preventing ID stealth, by explicitly checking whether the most voted servent (in the case of our example, B) is actually offering a resource with that digest.

Note that another version of ID stealth attack could be for D to steal the hash of a good resource r (in addition to B's reputation) in order to offer (if chosen for download) a bad resource. This spamming attack is however ineffective and even unlikely as the hash of the downloaded resources will not match r's one and the bad resource will therefore be immediately discarded (Phase 5 of Figure 1).

Shilling. Shilling is a well-known problem of distributed auctions protocols, where a malicious auctioneer may try to cheat his bidders by pushing up the price creating multiple registration names (called *shills*) to bid under. Shilling is different from pseudospoofing, as the multiple identities are actually created with real IP addresses, and not just simulated, by the attacker. In our setting, shilling could be used to create multiple witnesses in order to influence the voting procedure on a doctored resource or on a malicious servent. If enough apparently well-intentioned peers contribute to the resource's or the servent's reputation, the reputation may be pushed up enough to deceive a user into downloading a malicious resource. Our mechanism deals with shilling trying to ensure a high number of voters, making it expensive for the malicious user to create and maintain a sufficient number of shills. Also, shills are usually not involved in transactions other than the ones involving the malicious peer that created them; therefore, their own reputation is unlikely to be high. Note that our `TrueVote`/`TrueVoteReply` message exchange (Phase 3 of Figure 1) is aimed at forcing the malicious peer to actually allocate a different IP address for each shill.

[6]Note that the same result can be obtained even when A is behind a firewall, using the *push* variant of the Gnutella protocol.
[7]The term was first proposed by L. Detweiler on the Cypherpunks mailing list.

5. COMBINING SERVENT AND RESOURCE-BASED REPUTATIONS

XRep is fundamentally different from other reputation management systems for P2P [8] because it combines resources' and servents' reputations. Combining servent-based and resource-based reputations looks promising, as both schemes exhibit some shortcomings and advantages. A basic advantage of resource-based reputation systems is that votes actually express a property of the resource and not of its offerer; therefore they can be seen as more reliable and can carry more semantics. On the other hand, resource-based solutions can only be applied when resources have a history, that is the same resource is known to several servents. By contrast, in scenarios like eBay, where most resources appear in the system only once, it is important to be able to assign trust to servents.

By considering both resources and servents reputations we take the advantages of both approaches with respect to the following aspects.

- *Reputations' life cycle.* Thanks to servent-based reputation, new resources offered by a well established servent will be regarded as reliable. Beside making good behavior profitable, this has the advantage of avoiding cold-start problems for new resources (intuitively, a servent reputation can propagate to a potentially unlimited number of resources). On the other hand, resource-based reputations have a potentially wider scope and a longer life cycle, as a good resource will always be recognizable as such regardless of who offers it.

- *Impact on peers anonymity.* While servent-based reputation does not necessarily compromise anonymity, as reputations may be linked to pseudonyms, its effective use requires such pseudonyms to remain persistent. By contrast, resource-based reputations can be used in systems where peers can take on a fresh identity at every interaction.

- *Cold-start.* Servent-based reputations suffer from a cold-start problem in the sense that newcomers or peers offering a limited number of resources will struggle to actively participate in the distribution. Thanks to our integration of resource-based reputations, newcomers can immediately participate in distribution of well known resources.

- *Performance bottlenecks.* A servent-based reputation approach may foster bottleneck creation directing all downloads to the most reputable servents. Our technique for integrating resource-based reputations fosters load balancing as a resource can be safely downloaded from any of the servents offering it, allowing the choice of the best one for performance.

- *Blacklisting.* Resource-based reputations do not support blacklisting as no linking is made between a bad resource and the servent that provided it. Resource blacklisting, while ineffective as far as pinpointing malicious users is concerned, could be useful in stopping dissemination of malicious software. On the other hand, servent-based reputations can effectively support blacklisting only giving up on anonymity [7]. Pseudonyms blacklisting is only effective inasmuch they are persistent. In our context, since users may change pseudonym at no cost, blacklisting of malicious peers is expected to have a limited effectiveness as bad reputations can be nullified by taking on a fresh identity. However, a fresh identity will have no reputation at all, and therefore servents carrying it are unlikely to be selected.

- *Data storage and bandwidth requirements.* Our integrated approach deals with resources and servents reputations via the same polling algorithm. However, resource-based reputation systems require substantially more information to be maintained as experimental data [19] show that resources substantially outnumber servents.

- *Threshold effect.* Reputation-based systems can be exploited only if a reasonable threshold of votes can be achieved for each poll. As far as resource reputations are concerned, reaching this threshold would be made difficult by a uniform distribution of requests. In other words, no single resource could be widespread enough to allow for a sufficient number of voters, even making the assumption that all peers that have tried the resource would vote on it. However, our experiments (cf. Section 6) show that this is not a problem for a good percentage of resources. In a servent-based scenario reaching the threshold can be made more difficult by short life cycles of servent identities.

6. DISTRIBUTION OF SERVENTS AND RESOURCES IN GNUTELLA NETWORKS

An aspect that may have an impact on the success of XRep protocol is the distributions of servents and resources in the Gnutella network. It is reasonable to expect that servents and resources will be distributed non-uniformly, with i) few servents offering many resources, ii) many servents offering few resources, iii) few resources offered by many servents, and iv) many resources offered by few servents.

In particular we were expecting a Zipf (or, more generally, power-law) distribution, since this is the result that has been produced by many experimental studies in similar contexts. The results we obtained confirmed our expectations and can be the basis for the construction of an evaluation model that should provide numerical estimates on the effectiveness of the protocol.

An open source Gnutella client was modified and all the `QueryHit` and `Pong` messages traveling along the network were logged. As discussed in Section 2, the `QueryHit` message is generated by servents when they have in their repository a resource that satisfies the criteria in the `Query` message. The `Pong` message describes the number of files shared and their cumulative size. It is generated as an answer to `Ping` messages that are generated by servents when they connect to the network. `Pong` messages let users know the size of the portion of the network within their reach and the total number and size of resources available.

We logged half a million `QueryHit` records. The first result of the analysis of the logged records is presented in Figure 2. Resources are ordered by the number of their copies available on the network. Almost half of the 500.000 records were immediately discarded, as they represented multiple responses generated by the same servent for the same re-

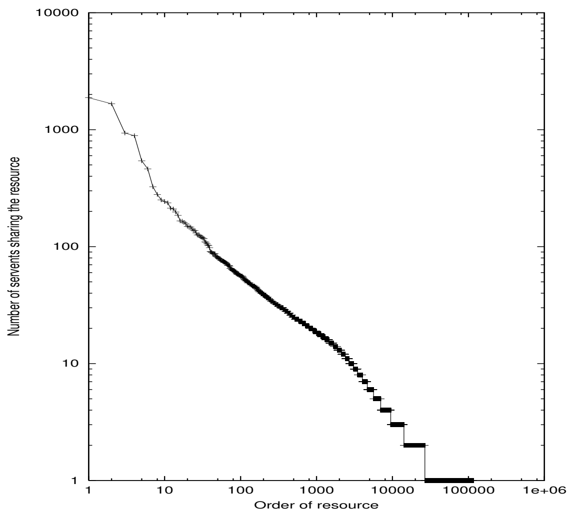

Figure 2: Resource distribution: number of servents sharing each resource

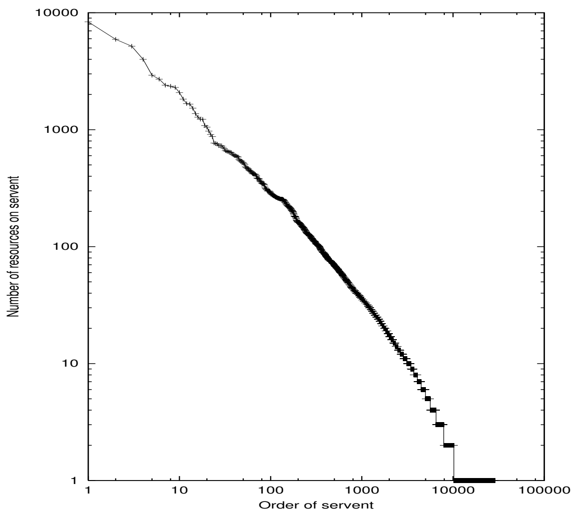

Figure 3: Servent distribution: number of resources on servents

source. We considered the name and the size of a resource to evaluate when a resource was identical; resources with identical semantic content, but with different size or name were considered distinct. The $253,712$ remaining messages contained a description of $116,219$ distinct resources; $89,485$ of them were offered by a single node. The graph in Figure 2 uses logarithmic scales for both the axes. When a power law distribution is presented in a log-log graph, a straight line appears; such a line is well recognizable in the graph, confirming our expectations.

Then, we used our log to estimate the concentration of resources on servents. We tried to estimate this distribution using the number reported in each `Pong` message, but the curve we obtained was quite irregular and unlikely to be a correct representation of the system behavior (anomalies in the information of `Pong` messages are also cited in [19]). Thus, we used again the `QueryHit` messages to evaluate resources' concentration. The results of this analysis are reported in Figure 3, showing the number of resources present on each servent. Servents are ordered according to the number of resources they offer. As one might expect, this distribution also fits nicely into the power law distribution; in particular, the alpha coefficient which characterizes the power law is 1.14, very close to value 1 that characterizes Zipf distributions.

Skewed distributions may appear at first to be detrimental to the applicability of our protocol, as for many resources or servents it will be difficult to obtain a number of responses to the poll able to guarantee a reputation. We argue instead that skewness is an advantage to the protocol.

If we consider the number of copies of a resource as an indicator of the level of interest for the resource by the user community–as it appears reasonable, otherwise, why should many servents offering it?–we may assume that frequent resources are more frequently searched; for each of these resources the number of votes will be high, even if we consider the portion of the P2P network within the horizon. The same line of reasoning applies to servents: servents offering many resources may be few, but they will answer with greater frequency to queries and will probably be well known by network participants. If the distributions of servents and resources are independent, the effectiveness of the combined resource/servent protocol will be the probabilistic sum of the effectiveness of the separate techniques.

We used our experimental observations to evaluate the presence of free-riders. All P2P clients can disable sharing and many users opt to do so. We estimated the rate of free-riders considering the percentage of `Pong` messages that showed no exported resources. The rate of free-riders we observed, 40.6%, lies between the value of 70% reported in [3] and the 25% value reported in [19]. Free riders are important because they are a potential source of votes that may, given the above result, almost duplicate the number of participants in the `XRep` protocol. A base assumption that we make is that each servent votes for itself and for the resources it offers. Then, a vote on a servent could be expressed by any node that has had the opportunity to download a resource from that servent; a vote on a resource could be expressed by any node that has had the opportunity to access the resource, even if the node does not share it because it had to remove the resource or because it is a free-rider.

The percentage of free-riders we detected is certainly high, but it is not certainly an indicator of a universal selfish attitude; indeed, 60% of the users share resources and offer them on the network, even if they do not obtain an immediate benefit from their actions. We feel confident to assume that users sharing resources would participate in our polling mechanism. For free-riders it is more difficult to foresee the level of participation; a full analysis of the motivations of free-riders would probably require a social and psychological investigation, more than a technical analysis. In our opinion there are currently two main motivations to the free-rider behavior: the first is the desire to avoid the costs, in terms of bandwidth use or general system load, that sharing involves; the other is the fear to be held accountable for the sharing of illegally duplicated copyrighted material. The participation to our protocol has a small impact on the above factors. First, the bandwidth or system load required from participants is small compared with the size and bandwidth required for the storage and transmission of the large files that are the typical content shared on these networks. Second, the level of exposure gained by voting on servent or resource identifiers is considerably smaller than that derived by the direct sharing of resources. These considerations lead to our perception that a significant portion of free-riders would participate in our polling protocol.

7. RELATED WORK

Several researchers have recently addressed the problem of enforcing security in the peer-to-peer scenario. One main line of work in the security community has been devoted to the enhancement of access control approaches with new authentication and authorization capabilities to address the fact that access requests may represent interactions between parties that know little about each other [5, 6, 11, 12, 14, 23]. All these works focused on allowing a peer acting as a server to restrict others' ability to access its resources. Peer-to-peer systems, however, also introduce other problems that reverse the security assumptions of traditional access control and require to focus the attention on providing protection *from those who offer* resources (servers), rather than from those who want to access them (clients). This paradigm shift is due to the inherent vulnerability of peer-to-peer systems from providers abusing the network to widespread tampered with resources.

Proposals to prevent or discouraging peers from distributing invalid or malicious content into the network are based on two main techniques: *micropayment-* and *reputation-*based trust systems [17].

Micropayment techniques (e.g., Mojo Nation, `www.mojonation.net`) are less closely related to our approach as they require peers to offer something of value in exchange of their participation in the system. Therefore, they impose a cost on malicious peers, as to insert invalid content into the network they would first need to provide a certain amount of resources.

Reputation models allow the expression and reasoning about *trust* in a peer based on its past behavior [18] and interactions other peers have experienced with it. Recent approaches (e.g., [15]) propose to facilitate trust between unacquainted parties by offloading risk to a trusted third party, which acquires revenues by assuming this risk. Of course, this technique requires dropping the complete uniformity assumption typical of P2P networks. A similar approach involving a reputation authority has been adopted by a number of projects. For instance, OpenPrivacy (`www.openprivacy.org`) introduces a set of *reputation services* that can be used to create, use, and calculate results from accumulated opinions and reputations. Sierra, Talon, and Reptile are OpenPrivacy projects that incorporate reputations to enhance searching as well as to discard unwanted information. Reputations are also effectively used in electronic marketplaces as a measure of the reliability of participants [24]. For instance, in eBay (`www.ebay.com`) each participant in a transaction can express a vote (`-1`, `0`, or `1`) on its counterparts. Votes so collected are used by eBay to provide cumulative ratings of users that are made known to all participants. In systems like eBay, reputations are associated with physical identities and are centrally managed at the eBay server. More in line with the peer-to-peer paradigm, several proposals (e.g., Poblano [7]) worked around the notion of *web of trust* where trust relationships and reputations are distributed and locally managed by each participant. The common ancestor of such approaches is probably PGP (`www.pgpi.org`), that allows users to certify other users' public keys without need for a Certification Authority. Some systems, such as GNUnet (`www.gnu.org/software/GNUnet/`) do not define globally scoped reputations, rather they rely on local opinions held by each node on its peers. Such opinions are based on past history and are neither shared nor cooperatively updated. On the same line, systems such as Advogato (`www.advogato.org/trust-metric.html`), assume full knowledge of the degree of trust that each peer has on others, and compute the transitive trust based on the closure of the resulting labeled graph. A more realistic approach is obtained by assuming that, while no peer can have full knowledge on the network, each of them can record the trust it has in others on the basis of its own experience, and communicate this information to others. In this category, are the Free Haven system [10] and the P2PRep proposal [8], where each peer can keep track, for each pseudonym it has interacted with, of good and bad experiences it had. Before downloading resources, peers can call others as witnesses of the reliability of the prospective source. Beside recording the peers' behavior in direct interactions, these systems can keep track also of the peers' reliability as witnesses, thus allowing for properly weighting their judgment [8, 9]. Voters credibility information can be easily taken into account in our XRep protocol by requiring voters to declare their identity and sign their votes as in [8]. Other proposals on the same line distinguish reliability of peers depending on the specific context of interaction [1, 22]. While Free Haven and P2PRep assume mainly the usage of positive reputations other proposals rely on the explicit use of negative reputations. This is the case for instance of the proposal by Aberer et al. [2], whose model assumes peers to be usually "honest" and considers therefore only dishonest interactions as relevant. After each transaction, and only in case of malicious behaviors, peers may file a complaint. Before engaging in interactions with others, peers can inquire the network about existing complaints on their counterparts. Full support of negative reputations, however, can only be achieved at the price of sacrificing anonymity [13]. Also, support of negative reputations only is risky as failure to retrieve existing complaints may result in trusting unreliable peers. While all these systems assume reputations to be associated with servents, the approach presented in this paper nicely combines servents and resource reputations.

8. CONCLUSIONS

Legal troubles involving P2P applications may give the feeling that P2P is another buzzword with a short life. Probably the hype will calm down, but P2P promises to be an important complement to current Web technologies. Overall, it may permit the construction of a specialized, yet easy to use, infrastructure for information sharing. However, it is a fact that its current success is mostly motivated by anonymous access to copyrighted material. This is indeed a thorny issue, and while we certainly do not support any illegal behavior, we acknowledge that a lively debate is currently ongoing. In this paper we dealt with technical aspects related to the support of anonymous and secure services. Our opinion is that proposals like ours, preserving anonymity to a degree, will probably evolve and may become a solution for quick-and-easy sharing of resources. This work represents a step towards a self-regulating P2P system which can also help in isolating from the network those nodes judged of illegal or unethical behavior. We are currently developing our research to provide XRep extensions to supernodes-based architectures and to define an enrichment of our votes and reputation value semantics.

9. ACKNOWLEDGMENTS

The work reported in this paper was partially supported by the Italian CNR Technologies and Services for Enhanced Content Delivery project and by the Roadmap for Advanced Research in Privacy Identity Management (RAPID) project under contract IST-2001-38310.

10. REFERENCES

[1] A. Abdul-Rahman and S. Hailes. Supporting trust in virtual communities. In *Proc. of the Hawaii International Conference on System Sciences*, Maui, Hawaii, January 2000.

[2] K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *Proc. of the Tenth International Conference on Information and Knowledge Management (CIKM 2001)*, Atlanta, Georgia, November 2001.

[3] E. Adar and B. Huberman. Free riding on gnutella. Technical report, Xerox PARC, August 2000.

[4] P.C. van Oorschot A.J. Menezes and S.A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[5] M. Blaze, J. Feigenbaum, J. Ioannidis, and A.D. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming: Issues in Distributed and Mobile Object Systems.* Springer Verlag – LNCS State-of-the-Art series, 1998.

[6] P. Bonatti and P. Samarati. Regulating service access and information release on the web. In *Proc. of the Seventh ACM Conference on Computer and Communications Security*, Athens, Greece, 2000.

[7] R. Chen and W. Yeager. Poblano - a distributed trust model for peer-to-peer networks. JXTA Security Project White Paper, 2001.

[8] F. Cornelli, E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Choosing reputable servents in a P2P network. In *Proc. of the Eleventh International World Wide Web Conference*, Honolulu, Hawaii, May 2002.

[9] C. Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *Proc. of the 2nd ACM Conference on Electronic Commerce*, Minneapolis, MN, USA, October 2000.

[10] R. Dingledine, M.J. Freedman, and D. Molnar. The Free Haven project: Distributed anonymous storage service. In *Proc. of the Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, California, USA, July 2000.

[11] V. Doshi, A. Fayad, S. Jajodia, and R. MacLean. Using attribute certificates with mobile policies in electronic commerce applications. In *Proc. of the 16th Annual Computer Security Applications Conference (ACSAC'00)*, pages 298–307, New Orleans, LA, 2000.

[12] C. Ellison. SPKI certificate documentation. http://www.pobox.com/~cme/html/spki.html.

[13] E.J. Friedman and P. Resnick. The social cost of cheap pseudonyms. *Journal of Economics and Management Strategy*, 10(2):173–199, Summer 2001.

[14] B. Gladman, C. Ellison, and N. Bohm. Digital signatures, certificates and electronic commerce. http://citeseer.nj.nec.com/277887.html.

[15] B. Horne, B. Pinkas, and T. Sander. Escrow services and incentives in peer-to-peer networks. In *Proc. of the 3rd ACM Conference on Electronic Commerce*, Tampa, Florida, USA, October 2001.

[16] KaZaA. http://www.kazaa.com.

[17] A. Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies.* O'Reilly & Associates, March 2001.

[18] P. Resnick, R. Zeckhauser, E. Friedman, and K. Kuwabara. Reputation systems. *Communications of the ACM*, 43(12):45–48, December 2000.

[19] S. Saroiu, P.K. Gummadi, and S.D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of Multimedia Computing and Networking (MMCN)*, San Jose, CA, USA, January 2002.

[20] J.B. Schafer, J.A. Konstan, and J. Riedl. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1/2):115–153, January 2001.

[21] *The Gnutella Protocol Specification v0.4 (Document Revision 1.2)*, June 2001. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[22] B. Yu and M.P. Singh. A social mechanism for reputation management in electronic communities. In *Proc. of the 4th International Workshop on Cooperative Information Agents (CIA)*, Boston, MA, USA, July 2000.

[23] T. Yu, M. Winslett, and K. Seamons. Interoperable strategies in automated trust negotiation. In *Proc. of 8th ACM Computer and Communication Security*, Philadephia, PA, November 2001.

[24] G. Zacharia, A. Moukas, and P. Maes. Collaborative reputation mechanisms in electronic marketplaces. In *Proc. of the 32nd Hawaii International Conference on System Sciences*, Maui, Hawaii, January 1999.