Managing and Sharing Servents' Reputations in P2P Systems

Ernesto Damiani, *Member*, *IEEE*, Sabrina De Capitani di Vimercati, *Member*, *IEEE Computer Society*, Stefano Paraboschi, and Pierangela Samarati, *Member*, *IEEE Computer Society*

Abstract—Peer-to-peer information sharing environments are increasingly gaining acceptance on the Internet as they provide an infrastructure in which the desired information can be located and downloaded while preserving the anonymity of both requestors and providers. As recent experience with P2P environments such as Gnutella shows, anonymity opens the door to possible misuses and abuses by resource providers exploiting the network as a way to spread tampered-with resources, including malicious programs, such as Trojan Horses and viruses. In this paper, we propose an approach to P2P security where servents can keep track, and share with others, information about the reputation of their peers. Reputation sharing is based on a distributed polling algorithm by which resource requestors can assess the reliability of perspective providers before initiating the download. The approach complements existing P2P protocols and has a limited impact on current implementations. Furthermore, it keeps the current level of anonymity of requestors and providers, as well as that of the parties sharing their view on others' reputations.

Index Terms—P2P network, reputation, credibility, polling protocol.

1 Introduction

In the world of Internet technologies, peer-to-peer (P2P) solutions are currently receiving considerable interest [14]. The term *peer-to-peer* is a generic label assigned to network architectures where all the nodes offer the same services and follow the same behavior. In Internet jargon, the P2P label represents a family of systems where the users of the network overcome the passive role typical of Web navigation and acquire an active role offering their own resources. P2P communication software is increasingly being used to allow individual hosts to anonymously share and distribute various types of information over the Internet [24]. While systems based on central indexes such as Napster (www.napster.com) collapsed due to litigations over potential copyright infringements, the success of "pure" P2P products like Gnutella [32] and Freenet [10] fostered interest in defining a global P2P infrastructure for information sharing and distribution. Several academic and industrial researchers are currently involved in attempts to develop a common platform for P2P applications and protocols [12], [19], [26]. Still, there are several thorny issues surrounding research on P2P architectures [5].

First of all, popular perception still sees P2P tools as a way to trade all kinds of digital media, possibly without the permission of copyright owners, and the legacy of early underground use of P2P networks is preventing the full acceptance of P2P technologies in the corporate world. Indeed, P2P systems are currently under attack by organizations like the RIAA (Recording Industry Association of America) and MPAA (Motion Picture Association of America), which intend to protect their intellectual property rights that they see violated by the exchange of copyrighted materials permitted by P2P systems. This opposition is testified by the lawsuits filed against P2P software distributors by the RIAA and MPAA. Of course, with this work, we do not intend to support the abuse of intellectual property rights. Our interest arises from the observation that P2P solutions are seeing an extraordinary success, and we feel that a self-regulating approach may be a way to make these architectures compliant with the ethics of the user population and isolate from the network the nodes offering resources that are deemed inappropriate by the users.

Second, a widespread security concern is due to the complete lack of peers' accountability on shared content. Most P2P systems protect peers' anonymity allowing them to use self-appointed *opaque identifiers* when advertising shared information (though they require peers to disclose their IP address when downloading). Also, current P2P systems neither have a central server requiring registration nor keep track of the peers' network addresses. The result of this approach is a kind of *weak anonymity*, that does not fully avoid the risks of disclosing the peers' IP addresses, prevents the use of conventional *Web of trust* techniques [21], and allows malicious users to exploit the P2P infrastructure to freely distribute Trojan Horses and viruses. Some practitioners contend that P2P users are no more exposed to viruses than when downloading files from the Internet through conventional means such as FTP and the Web, and that virus scanners can be used to prevent infection from digital media downloaded from a P2P network. However, using P2P

- *E. Damiani, S. De Capitani di Vimercati, and P. Samarati are with the Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, Via Bramante 65, 26013 Crema, Italy. E-mail: {damiani, decapita, samarati}@dti.unimi.it.*
- *S. Paraboschi is with the Dipartimento di Ingegneria, Università di Bergamo, Via Marconi 5, 24044 Dalmine, Italy. E-mail: parabosc@unibg.it.*

Manuscript received 15 July 2002; revised 15 Dec. 2002; accepted 6 Jan. 2003. For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 118231.

software undeniably increases the chances of being exposed, especially for home users who cannot rely on a security policy specifying which antivirus program to use and how often to update it. Moreover, with FTP and the Web, users most typically execute downloaded programs only when they trust the site where the programs have been downloaded from. We believe that the future development of P2P systems will largely depend on the availability of novel provisions for ensuring that peers obtain reliable information on the quality of the resources they are retrieving. In the P2P scenario, such information can be obtained by means of *peer review*, that is, relying on the peers' opinions to establish a *digital reputation* for information sources on the P2P network.

In this paper, we propose an approach for managing and sharing peer reputations in a P2P network. We focus on pure P2P networks for file exchange and, more precisely, on the Gnutella architecture [32]. The reason for focusing on a pure P2P network is that it is closest to the ideal structure of the peer-to-peer spirit, where all participants have a uniform role. Also, our solution builds on Gnutella in particular because it is an open and consolidated protocol and many open source implementations are available that permit to experiment with our protocol variants. However, we note that our solution can be also adopted within other environments where indexing schemes are present, such as Chord [30].

Our digital reputations can be seen as the P2P counterparts of client-server digital certificates [15], [18], but present two major differences that require them to be maintained and processed very differently. First of all, reputations must be associated with self-appointed opaque identifiers rather than with externally obtained identities. Therefore, keeping a stable identifier (and its good reputation) through several transactions must provide a considerable benefit for peers' wishing to contribute information to the network, while continuously reacquiring newcomer status must not be too much of an advantage for malicious users changing their identifier in order to avoid the effect of a bad reputation. Second, while digital certificates have a long life-cycle, the semantics of the digital reputation must allow for easily and consistently updating them at each interaction. In our approach, reputations simply certify the experience accumulated by other peers' when interacting with an information source and smoothly evolve over time via a polling procedure.

2 Sketch of the Approach

We first define the working of the Gnutella protocol and then sketch the basic idea of our approach that extends the Gnutella protocol to support reputation establishment and sharing.

2.1 Gnutella Overview

Gnutella offers a fully peer-to-peer decentralized infrastructure for information sharing. Each servent is associated with a self-appointed *servent_id*, which can be communicated to others when interacting, as established by the P2P communication protocol used. The servent_id of a party (intuitively a user connected at a machine) can change at any instantiation or remain persistent. The topology of a Gnutella network graph is meshed, and all servents act both as clients and servers and as routers propagating incoming messages to neighbors. While the total number of nodes of a network is virtually unlimited, each node is linked dynamically to a small number of neighbors, usually between 2 and 12. Messages, that can be broadcast or unicast, are labeled by a unique identifier and can be used by the recipient to detect where the message comes from. This feature allows replies to broadcast messages to be unicast when needed. To reduce network congestion, all the packets exchanged on the network are characterized by a given *Time-To-Live* (TTL). 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 limit of the TTL creates a *horizon* of visibility for each node on the network. The horizon is defined as the set of nodes residing on the network graph at a path length equal to the TTL and reduces the scope of searches, which are therefore forced to work on only a portion of the resources globally offered. A node's horizon depends on the number of connections that it opens with its neighbors (typical values are in the range 2 to 6). For each Gnutella node, the number of reachable peers increases polynomially with the number of connections and exponentially with the value of TTL; nodes may dynamically reduce the values of TTL and connections when they detect congestion.

A P2P file exchange in Gnutella involves two phases: *search* and *download* (see Fig. 1). To search for a particular file, a servent p sends a broadcast `Query` message to every node to which it is directly linked. Since the message is broadcast through the P2P network, each node not directly connected with p will receive this message via intermediaries (and will not know that p originated it). Servents that receive the query and have in their repository the file requested, answer with a `QueryHit` unicast packet that contains a *ResultSet* plus their IP address and the port number of a server process from which the files can be downloaded using the HTTP protocol. Although p is not known to the responders, responses can reach p via the network by following in reverse the same connection arcs used by the query. Among the servers responding to the query, p can select the servent from which to execute the download. This choice is usually based on the offer quality (e.g., the number of hits and the declared connection speed) or on preference criteria based on its past experiences. The download is carried out via a direct connection based on protocols on the TCP/IP family (FTP, HTTP).

Servents can gain a complete vision of the network within the horizon by broadcasting `Ping` messages. Servents within the horizon reply with a `Pong` message containing the number and size of the files they share. Finally, communication with servents located behind firewalls is ensured by means of `Push` messages. A `Push` message behaves more or less like passive communication in traditional protocols such as `FTP`, in as much it requires the "pushed" servent to initiate the connection for downloading.

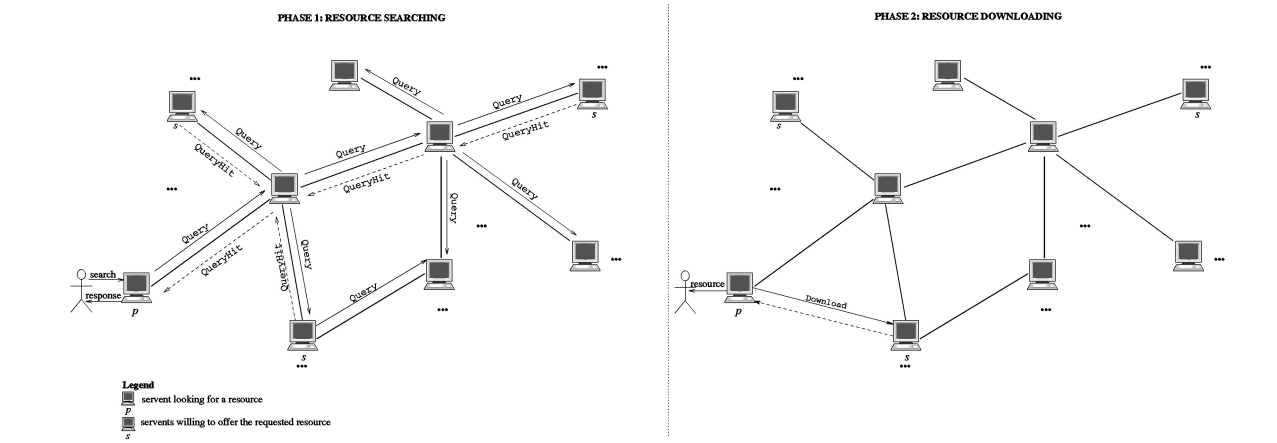

Fig. 1. Gnutella working.

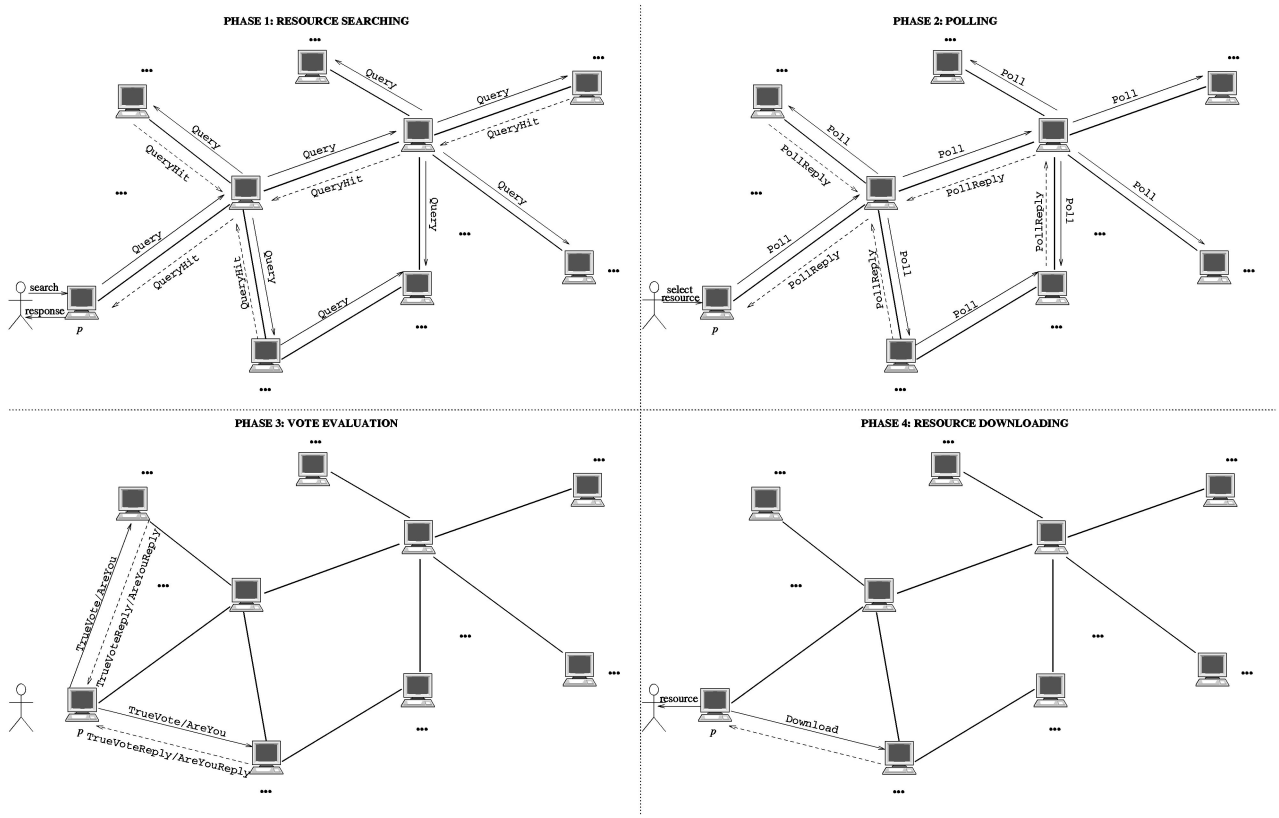

Fig. 2. `P2PRep` protocol working.

2.2 Basic Idea of `P2PRep`

The basic idea of our approach, called `P2PRep`, is to allow p, before deciding from where to download the resource, to enquire about the reputation of offerers by polling its peers. The approach complements the Gnutella protocol with two phases: *polling* and *vote evaluation* (phases 2 and 3 in Fig. 2). After receiving the responses to its query, p can select a servent (or a set of servents) based on the quality of the offer and its own past experience. Then, p polls its peers by broadcasting a (`Poll`) message requesting their opinion about the selected servents. All peers can respond (`Poll-Reply`) to the poll with their opinions about the reputation of each of such servents. The poller p can use the opinions expressed by these *voters* to make its decision.

The intuitive idea behind our approach is therefore very simple. A little complication is introduced by the need to prevent exposure of polling to security violations by malicious peers. In particular, we need to ensure authenticity of servents acting as offerers or voters (i.e., preventing impersonation) and the quality of the poll, ensuring the votes and detecting possible dummy votes expressed by servents acting as a clique under the control of a single malicious party. Also, our approach encourages persistence of the servent identities as the only way to maintain history of a servent_id across transactions. However, persistence of

a servent_id does not affect anonymity of the party behind it, as the servent_id works only as an opaque identifier.[1]

We present two flavors of reputation sharing. In the first solution, which we call *basic polling*, the servents responding to the poll do not provide their servent_id. In the second solution, which we call *enhanced polling*, voters also declare their servent_id, which can then be taken into account by p for weighting the votes received (p can judge some voters as being more credible than others).

In the next section, we illustrate how the different phases *resource searching*, *polling*, *vote evaluation*, and *resource downloading* sketches in Fig. 2 are carried out in our protocols.

3 Reputation-Based Source Selection Protocols

Hashing and cryptographic techniques are used to provide the basic security functionalities to our protocols. In particular, we assume that poll responses have included an integrity check computed via a secure hash function and that direct transmissions (i.e., outside the P2P network) are carried out on a secure channel (e.g., SSL). Also, our protocols assume the use of public key encryption to provide integrity and confidentiality of message exchanges. Whether permanent or fresh at each interaction, we require each servent_id to be a digest of a public key, obtained using a secure hash function and for which the servent knows the corresponding private key. This assumption allows a peer talking to a servent_id to ensure that its counterpart knows the private key, whose corresponding public key, the servent_id, is a digest. A pair of keys is also generated on the fly for each poll. In the following, we will use $(\mathrm{PK}_i, \mathrm{SK}_i)$ to denote a pair of public and private keys associated with i, where i can be a servent or a poll request. We will use $\mathrm{E}_K(M)$ and $\mathrm{S}_K(M)$ to denote the encryption and signature, respectively, of message M under key K, and $h(M)$ to denote the hashing of message M. Also, in illustrating the protocols, we will use p to denote the protocol's initiator, O to denote the set of servents responding to the query (*offerers*), and V to denote the set of servents responding to p's polling (*voters*).

3.1 Basic Polling Protocol

Fig. 3 illustrates the steps composing the different phases in the basic polling solution. In the figure, a "(**G**)" associated with a step indicates that the step pertains to traditional Gnutella interchange; unmarked steps are peculiar to our protocol. The protocol works as follows:

Phase 1: Resource searching. This phase works in the same way as in the conventional Gnutella protocol. Servent p looking for a resource broadcasts a `Query` indicating the resource it is looking for. Every servent receiving the query and willing to offer the requested resource for download, sends back a `QueryHit` message stating how it satisfies the query (i.e., number of query hits, the set of responses, and the speed in Kb/second) and providing its servent_id and its pair $\langle IP, port \rangle$, which p can use for downloading.

Phase 2: Polling. Upon reception of the `QueryHit` messages, p selects a top list of favorite servents T and polls its peers about the reputations of these servents. In the poll request, p includes the set T of servent_ids about which it is enquiring and a public key PK_{poll} generated on the fly for the poll request, with which responses to the poll will need to be encrypted.[2] The poll request is sent through the P2P network and, therefore, p does not need to disclose its servent_id or its IP to be able to receive back the response. Peers receiving the poll request and wishing to express their opinion on any of the servants in the list, send back a `PollReply` expressing their votes and pair $\langle \mathrm{IP}, \mathrm{port} \rangle$ pair (like when responding to queries). A hash of the votes and pair $\langle \mathrm{IP}, \mathrm{port} \rangle$ is also added in order to allow p to check the integrity of the message. The `PollReply` is then encrypted with PK_{poll} to ensure its confidentiality (of both the vote and the voters) when in transit.

Phase 3: Vote evaluation. As a result of the previous phase, p receives a set of votes, where, for each servent in T, some votes can express a good opinion while some others can express a bad opinion. To base its decision on the votes received, p needs to trust the reliability of the votes. Thus, p first uses the hash to detect tampered-with votes and discard them. Second, p detects votes that appear suspicious, for example, since they are coming from IPs suspected of representing a clique.[3] Third, p selects a set of voters that it directly contacts (by using the $\langle \mathrm{IP}, \mathrm{port} \rangle$ pair they provided) to check whether they actually expressed that vote. For each selected voter v_i, p directly sends a `TrueVote` request reporting the votes it has received from v_i, and expects back a confirmation message `TrueVoteReply` from v_i confirming the validity of the vote. This forces potential malicious servents to pay the cost of using IPs as false witnesses. Note that, of course, nothing forbids malicious servents to completely throw away the votes in transit (but if so, they could have done this blocking on the `QueryHit` in the first place). Also, note that servents will not be able to selectively discard votes, as their recipient is not known and their content, being encrypted with PK_{poll}, is not visible to them. Upon assessing correctness of the votes received, p can finally select the offerer it judges as its best choice, according to the 1) connection speed, 2) its own reputation about the servents, and 3) the votes received. Different criteria can be adopted for evaluating the votes received, and any servent can use its own. For instance, p can choose the offerer with the highest number of positive votes, the one with the highest number of positive votes among the ones for which no negative vote was received, the one with the higher difference between the number of positive and negative votes, and so on.

Phase 4: Resource downloading. At this point, before actually initiating the download, p challenges the selected offerer s to assess whether it corresponds to the declared servent_id. Servent s will need to respond with a message containing its public key PK_s and the challenge signed with its private key SK_s. If the challenge-response exchange succeeds and the PK_s's digest corresponds to the servent_id

1. It must be noted that, while not compromising anonymity, persistent identifiers introduce linkability, meaning transactions coming from a same servent can be related to each other.

2. In principle, p's key could be used for this purpose, but this choice would disclose the fact that the request is coming from p.

3. We will elaborate more on this in Section 3.4.

Protocol 1 *Basic Polling protocol*

Initiator: Servent p

Peers: Participants in the message broadcasting, among which a set O of offerers and a set V of voters

INITIATOR

Phase 1: Resource searching

(**G**) 1.1 Start a search request by broadcasting a `Query` message

`Query`(*min_speed,search_string*)

(**G**) 1.2 Receive a set of offers from offerers O

`QueryHit`(*num_hits,port,IP,speed,Result,trailer,servent_id*$_i$)

Phase 2: Polling

2.1 Select top list $T \subseteq O$ of offerers

2.2 Generate a pair of public-secret keys ($\mathrm{PK}_{poll}, \mathrm{SK}_{poll}$)

2.3 Poll peers about the reputations of offerers T

`Poll`(T,PK_{poll})

2.4 Receive a set of votes from voters V

`PollReply`($E_{\mathrm{PK}_{poll}}$*(IP,port,Votes,h(IP,port,Votes))*)

Phase 3: Vote evaluation

3.1 Remove from V voters that appear suspicious (e.g., checking IP addresses)

3.2 Select a random set $V' \subseteq V$ of voters and for each $v_i \in V'$ check whether it actually expressed that vote

`TrueVote`(*Votes*$_i$)

3.3 Expect back confirmation messages from each selected voter $v_i \in V'$

`TrueVoteReply`(*response*$_i$)

Phase 4: Resource downloading

4.1 Select servent s from which download files

4.2 Generate a random string r

4.3 Send a `challenge` message to s

`challenge`(r)

4.4 Receive a `response` message from s containing its public key PK_s and the challenge signed with its private key SK_s

`response`($\mathrm{S}_{\mathrm{SK}_s}$(r),$\mathrm{PK}_s$)

4.5 If the challenge-response exchange fails terminate the process

(**G**) 4.6 Download the files

4.7 Update experience_repository

PEERS

Q.1 Upon receiving a search request (`Query` message), check if any locally stored files match the query and if so send a `QueryHit` message

Q.2 Broadcast the query through the P2P network

P.1 Upon receiving a `Poll` message, check if any of the servents listed in it are known and express an opinion on them by sending a `PollReply` message

P.2 Broadcast the `Poll` message through the P2P network

P.3 Upon receiving a `TrueVote` message, confirm the votes by sending a `TrueVoteReply` message

Fig. 3. Sequence of messages and operations in the basic polling protocol.

that s has declared, then p will know that it is actually talking to s. Note that the challenge-response exchange is done, like the download, via direct communication on a secure connection, to prevent impersonation by which

servents can offer resources using the servent_id of other peers. With the authenticity of the counterpart established, p can finally download the resource and, depending on its satisfaction for the download, update its reputation information for s (see Section 3.3).

3.2 Enhanced Polling Protocol

The enhanced polling protocol differs from the basic solution by requesting voters to provide their servent_id. Intuitively, while in the basic polling a servent only maintains a local recording of its peers reputation, in the enhanced solution, each servent also maintains track of the *credibility* of its peers, which it will use to properly weigh the votes they express when responding to a poll request. Fig. 4 illustrates the different steps composing the phases of the protocol. Like for the basic polling protocol, a "(**G**)" associated with a step indicates that the step pertains to traditional Gnutella interchange; a "(*)" associated with a step indicates that the step is different from the corresponding step in the basic polling.

Phase 1: Resource searching. This phase works like in the basic polling protocol.

Phase 2: Polling. Like for the basic protocol, after receiving the `QueryHit` responses and selecting its top list T of choice, p broadcasts a poll request enquiring its peers about the reputations of servents in T. A servent receiving the poll request and wishing to express an opinion on any of the servents in T can do so by responding to the poll with a `PollReply` message in which, unlike for the basic case, it also reports its servent_id. More precisely, `PollReply` reports, encrypted with PK_{poll}, the public key PK_i of the voter v_i and its vote declarations signed with the corresponding private key SK_i. The vote declaration contains the pair $\langle \mathrm{IP}, \mathrm{port} \rangle$ and the set of votes together with the servent_id of the voter. Once more, the fact that votes are encrypted with PK_{poll} protects their confidentiality and the signature allows the detection of integrity violations. In addition, the fact that votes are signed with the voter's private key guarantees the authenticity of their origin.

Phase 3: Vote evaluation. Again, after collecting all the replies to the poll, p carries out an analysis of the votes received removing suspicious votes. It then selects a set of newcomer[4] voters to be contacted directly to assess the correct origin of votes. This time, the direct contact is needed to avoid servent_id to declare fake IPs (there is no need to check the integrity of the vote as the vote's signature guarantees it). Selected voters are then directly contacted, via the $\langle \mathrm{IP}, \mathrm{port} \rangle$ pair they provided with an `AreYou` message reporting the servent_id that was associated with this pair in the vote. Upon this direct contact, the voter responds with an `AreYouReply` message confirming its servent_id. Servent p can now evaluate the votes received in order to select, within its top list T, the server it judges best. While in the basic polling, all votes were considered equal; the knowledge about the servent_ids of the voters allows p to weigh the votes received based on who expressed them. This distinction is based on credibility information maintained by p and reporting, for each servent s that p wishes to record, how much p trusts the opinions expressed by s (see Section 3.3).

4. Servents that have voted in past polls and for which an `AreYou` check was already executed and the $\langle IP, port \rangle$ recorded need not be verified.

Phase 4: Resource downloading. Like for the basic case, we assume that, before downloading, a challenge-response exchange is executed to assess the fact that the contacted servent s knows the private key SK_s such that the digest of the corresponding PK_s is the declared servent_id. After the downloading, and depending on the success of the download, p can update the reputation and credibility information it maintains (see Section 3.3).

3.3 Maintaining Servents' Reputations and Credibilities

When illustrating our protocols, we assumed that each servent maintains some information about how much it trusts others with respect to the resources they offer (*reputation*) and the votes they expressed in the past (*credibility*). Different approaches can be used to store, maintain, and express such information, as well as to translate it in terms of votes and vote evaluation. In fact, many vote aggregation systems [8] are available, each suited to a different set of applications. Some aggregation techniques are independent from the polling algorithm; others (e.g., the ones requiring multiple rounds) must be at least partially known to the voters in advance. In our protocol, we have chosen not to make voters aware of a set of alternatives (i.e., the alternative servents offering a resource), as this is known to introduce additional security weaknesses in the voting protocol [8]. Our algorithm proposes a single alternative per voting round and transparently aggregates results using a (clustered) compensative aggregation. Servent ranking is then computed aggregating cluster-wide results. In this section, we briefly illustrate the approach we adopted in our current implementation. Our technique includes three different steps, each involving a distinct *aggregation operator*. First, each voter aggregates stored values representing its past *experience* in order to decide the vote to cast about a given servent. Second, the poller aggregates values representing the reliability of past recommendations on the part of each voter to represent each voter's *credibility*. Third, the poller uses voters' credibilities to aggregate received votes for each servent, in order to compute the final servents' ranking and to pick the best source from which it can download the resource it needs.

Aggregating experience values into votes. Each servent s maintains its overall *experience_repository* as a set Ψ of triples $\psi = (servent\_id, num\_plus, num\_minus)$. Each triple ψ represents the history of past interactions with a given servent_id, reporting the number of successful (*num_plus*) and unsuccessful (*num_minus*) downloads that s experienced. Servent s could judge a download as unsuccessful, for example, if the downloaded resource was unreadable, corrupted, or it included malicious content. This experience_repository is updated after each download by incrementing the suitable counter, according to the download outcome. In our approach, each voter casts its vote by using its experience values to compute a binary outcome that can be either positive (`1`) or negative (`0`).[5] This outcome is based

5. While here we consider binary votes only, 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 peer reliable at 80 percent).

Initiator: Servent p

Peers: Participants in the message broadcasting, among which a set O of offerers and a set V of voters

INITIATOR

Phase 1: Resource searching

(**G**) 1.1 Start a search request by broadcasting a `Query` message

`Query`(min_speed,$search\_string$)

(**G**) 1.2 Receive a set of offers from offerers O

`QueryHit`(num_hits,$port$,IP,$speed$,$Result$,$trailer$,$servent\_id_i$)

Phase 2: Polling

2.1 Select top list $T \subseteq O$ of offerers

2.2 Generate a pair of public, secret keys $(\mathrm{PK}_{poll}, \mathrm{SK}_{poll})$

2.3 Poll peers about the reputations of offerers T

`Poll`(T,PK_{poll})

(*) 2.4 Receive a set of votes from voters V

(*) `PollReply`($E_{\mathrm{PK}_{poll}}$(IP,$port$,$Votes$,$servent\_id_i$,S_{SK_i}(IP,$port$,$Votes$, $servent\_id_i$),PK_i))

Phase 3: Vote evaluation

3.1 Remove from V voters that appear suspicious (e.g., checking IP addresses)

(*) 3.2 Select a random set $V' \subseteq V$ of voters and for each $v_i \in V'$ check its identity by sending an `AreYou` message

(*) `AreYou`($servent\_id_i$)

(*) 3.3 Expect back confirmation messages from each selected voter

(*) `AreYouReply`($response_i$)

Phase 4: Resource downloading

4.1 Select servent s from which download files

4.2 Generate a random string r

4.3 Send a `challenge` message to s

`challenge`(r)

4.4 Receive a `response` message from s containing its public key PK_s and the challenge signed with its private key SK_s

`response`($\mathrm{S}_{\mathrm{SK}_s}$(r),$\mathrm{PK}_s$)

4.5 If the challenge-response exchange fails terminate the process

(**G**) 4.6 Download the files

(*) 4.7 Update experience and credibility repositories

PEERS

Q.1 Upon receiving a search request (`Query` message), check if any locally stored files match the query and if so send a `QueryHit` message

Q.2 Broadcast the query through the P2P network

P.1 Upon receiving a `Poll` message, check if know any of the servents listed in it and express an opinion on them by sending a `PollReply` message

P.2 Broadcast the `Poll` message through the P2P network

(*) P.3 Upon receiving an `AreYou` message, confirm the identity by sending an `AreYouReply` message

Fig. 4. Sequence of messages and operations in the enhanced polling protocol.

on a suitable *aggregation operator* $\phi : \Psi \rightarrow \{0, 1\}$ that each voter adopts independently [16]. For instance, a peer may take a *conservative* approach and decide to vote positively only for servents with which it never had bad experiences (in this case, $\phi(\psi) = 1$ if $num\_minus = 0$, $\phi(\psi) = 0$ otherwise); on the other hand, other peers could adopt a more

compensatory attitude, balancing bad and good experiences as follows: $\phi(\psi) = 1$ if $num\_plus - num\_minus \geq 0$, $\phi(\psi) = 0$ otherwise.

Computing voters' credibility. Once received, votes need to be aggregated to compute a ranking of available sources. Of course, one could rank sources simply according to the number of votes each one gets; but, this approach would not make any distinction among voters that are known to be reliable and others that are unknown or, worse, that are known to have been unreliable in the past. To take this difference into account, we introduce the concept of voter's *credibility*. Each servent s maintains a *credibility_repository* as a set Θ of triples

$$\theta = (servent\_id, num\_agree, num\_disagree)$$

associating with each *servent_id* its accuracy in casting votes. In particular, *num_agree* represents the number of times the servent_id's opinion on another peer x (within a transaction in which x was selected by s for downloading) matched the outcome of the download. Conversely, *num_disagree* represents the number of times the servent_id's opinion on another peer x (again, within a transaction in which x was then selected by s for downloading) did not match the outcome of the download. A simple approach to credibility_repository maintenance is as follows: At the end of a successful transaction, the initiator p increases by one the *num_agree* counter of all those servents that had voted in favor of the selected servent x, and increases by one the *num_disagree* counter of all those servents that had voted against x. The vice versa happens for unsuccessful transactions. As in the previous step, our credibility computation is based on a simple *aggregation operator*. Such an operator may be binary, that is $\phi : \Theta \rightarrow \{0, 1\}$. Again, we can adopt a conservative attitude simply by choosing $\phi(\theta) = 1$ if $num\_agree - num\_disagree \geq k$, and $\phi(\theta) = 0$ otherwise, where k is a positive integer.[6] While a complete discussion on this topic is outside the scope of this paper, it is interesting to note that the value of k could be adaptively tuned for each voter in order to reflect changes in its reliability. Alternatively, we can define $\phi : \Theta \rightarrow [0, 1]$, for example, as $\phi = \frac{num\_agree}{num\_agree + num\_disagree}$. Note that, this time, the codomain of the ϕ function is the whole unit interval and not just the pair of values $\{0, 1\}$.

Aggregating votes to rank servents. The poller uses votes and, optionally, the voters' credibilities in order to compute a final ranking of sources. Once again, this aggregation can be performed using a number of techniques, corresponding to different attitudes on the part of the poller [4]. Basically, we can imagine that each vote is weighted by multiplying it with the credibility of the voter that casted it. Different aggregation techniques can provide a different choice for combining these weighted votes, including: *logical conjunction*, which considers the minimum among the weighted votes; *product-based conjunction*, which considers their product; or *weighted averages* that computes the average of the weighted votes.

6. Such binary aggregations are often referred to as "k times and you are out" operators.

3.4 Removing Suspects from the Poll

`PollReply` messages need to be verified in order to prevent malicious users from creating or forging a set of peers with the sole purpose of sending in positive votes to enhance their reputation. We base our verification on a *suspects identification* procedure, trying to reduce the impact of forged votes. Our procedure relies on computing clusters of voters whose common characteristics suggest that they may have been created by a single, possibly malicious, user. Of course, nothing can prevent a malicious user, aware of the clustering technique, from forging a set of voters all belonging to different clusters; this is however discouraged by the fact that some of the voting peers will be contacted in the following check phase (Step 3.2). In principle, voters' clustering can be done in a number of ways, based on application-level parameters, such as the branch of the Gnutella topology through which the votes were received, as well as on network-level parameters, such as IP addresses. At first sight, IP-address clustering based on `net_id` appears an attractive choice as it is extremely fast and does not require generating additional network traffic. An alternative, more robust, approach currently used by many tools, such as IP2LL [20] and NetGeo [23], computes IP clustering by accessing a local `Whois` database to obtain the IP *address block* that includes a given IP address. We are well aware that, even neglecting the effects of IP spoofing, both IP clustering techniques are far from perfect, especially when clients are behind proxies or firewalls, so that the "client" IP address may actually correspond to a proxy. For instance, the AOL network has a centralized cluster of proxies at one location for serving client hosts located all across the US, and the IP addresses of such clusters all belong to a single address block [25]. In other words, while a low number of clusters may suggest that a voters' set is suspicious, it does not provide conclusive evidence of forgery. For this reason, we do not use the number of clusters to conclude for or against voters' forgery; rather, we compute an aggregation (e.g., the arithmetic mean) of votes expressed by voters in the same cluster. Each cluster can then correspond to one or more votes (depending on its size). After the outcome has been computed, an explicit *IP checking phase* starts: a randomized sample of voters are contacted via direct connections, using their alleged IP addresses. If some voters are not found, the sample size is enlarged. If no voter can be found, the whole procedure is aborted.

4 `P2PRep` Impact on Gnutella-Like P2P Systems

The impact of `P2PRep` on a real-world P2P system based on Gnutella depends on several factors, some of them related to the original design of Gnutella itself. First of all, in the original design, there is no need for a servent to keep a persistent servent identifier across transactions; indeed, several Gnutella clients generate their identifiers randomly each time they are activated. `P2PRep` encourages servents keen on distributing information to preserve their identifiers, thus contributing to a cooperative increase of the P2P community's ethics. Second, efficiency considerations

brought Gnutella designers to impose a constraint on the network *horizon*, so that each servent only sees a small portion of the network. This influences P2PRep impact, since in real-world scenarios a poller may be able to get a reasonable number of votes only for servents that have a high rate of activity. In other words, P2PRep will act as an adaptive selection mechanism of reliable information providers within a given horizon, while preserving the "pure" P2P nature of a Gnutella network. Another major impact factor for P2PRep is related to performance, as Gnutella is already a verbose protocol [28] and the amount of additional messages required could discourage the use of P2PRep. However, the protocol operation can be easily tuned to the needs of congested network environments. For instance, in Section 3, we have assumed that peers express votes on others upon explicit polling request by a servent. Intuitively, we can refer to this polling approach as *client-based*, as peers keep track of good and bad experiences they had with each peer s they used as a source. In low-bandwidth networks, P2PRep message exchanges can be reduced by providing a server-based functionality, whereby servents keep a record of (positive) votes for them stated by others. We refer to these "reported" votes as *credentials*, which the servent can provide in the voting process. Obviously, credentials must be signed by the voter that expressed them, otherwise a servent could fake as many credentials as it likes. Finally, when a P2P system is used as a private infrastructure for information sharing (e.g., in corporate environments), P2PRep vote semantics can easily be tuned adopting a rating system for evaluating the quality of different information items provided by a servent, rather than its reliability or malicious attitude.

4.1 Security Improvements

Of course, the major impact of a reputation based protocol should be on improving the global security level. P2PRep has been designed in order to alleviate or resolve some of the current security problems of P2P systems like Gnutella [5]. Also, P2PRep tries to minimize the effects of some well-known weaknesses usually introduced by poll-based distributed algorithms. In this section, we discuss the behavior of our protocol with respect to known attacks. Throughout the section, we assume Alice to be a Gnutella user searching for a file, Bob to be a user who has the file Alice wants, Carl to be a user located behind a firewall who also has the file Alice wants, and David to be a malicious user.

Distribution of tampered-with information. The simplest version of this attack is based on the fact that there is virtually no way to verify the source or contents of a message. A particularly nasty attack is for David to simply respond providing a fake resource with the same name as the real resource Alice is looking for. The actual file could be a Trojan Horse program or a virus. Currently, this attack is particularly common as it requires virtually no hacking of the software client. Both the basic and enhanced version of our protocol are aimed at solving the problem of impersonation attacks. When Alice discovers that the resource she downloaded from David is faked, she will downgrade David's reputation, thus preventing further interaction with him. Also, Alice will become a material witness against David in all polling procedures called by others. Had David previously spent an effort to acquire a good reputation, he will now be forced to drop his identifier, reverting to newcomer status and dramatically reducing his probability of being chosen for future interactions.

Man in the middle. This kind of attack takes advantage of the fact that the malicious user David can be in the path between Alice and Bob (or Carl). The basic version of the attack goes as follows: First, Alice broadcasts a Query message and Bob responds. David intercepts the QueryHit message from Bob and rewrites it with his IP address and port instead of Bob's. Alice then receives David's reply. Now, if Alice decides to download the file from David, David can download the original content from Bob, infect it, and pass it on to Alice. A variant of this attack relies on push-request interception. In this case, Alice generates a Query message and Carl responds. Alice attempts to connect but Carl is firewalled, so she generates a Push message. David intercepts the Push message from Alice and forwards it with his IP address and port. Now, Carl connects to David and transfers his content and then David connects to Alice and provides the modified content.

While both flavors of this attack require substantial hacking of the client software, they are very effective, especially because they do not involve IP spoofing and, therefore, cannot be prevented by network security measures. Our protocols address these problems by including a challenge-response phase (Steps 4.2 through 4.5) just before downloading. In order to impersonate Bob (or Carl) in this phase, David should be able to design a pair of keys such that the digest of the public key is the Bob's identifier. Therefore, both versions of this attack are successfully prevented by our protocols.

5 Implementing P2PRep in the Gnutella Environment

We have implemented our protocol as an extension to an existing Gnutella system. In this section, we describe how the P2PRep protocol is implemented and the modifications it requires to a standard Gnutella servent's architecture.

5.1 P2PRep Messages

To keep the impact of our proposed extension to a minimum, all P2PRep messages are carried as payload inside ordinary Query and QueryHit messages. Fig. 5 shows the P2PRep messages and their structure; the numbers below the message's fields represent their size expressed in bytes. Specifically, Poll messages are implemented using the field *search_string* of standard Query messages. To identify the message and its encoding correctly, the *search_string* field contains the string REP:poll:HEX as shown in Fig. 5, followed by a list of the *servent_ids* of all servents whose reputation is being checked. At the end of the message, there is the public key of the polling session. Standard Gnutella servents will process Poll messages as ordinary queries that do not match any file.

In turn, PollReply messages are realized by using QueryHit messages. The QueryHit standard message includes *num_hits* triples of the form (*file size, file index, file name*) that compose the *Result set*. In a PollReply message, the first triple contains zeros both as size and index, and the

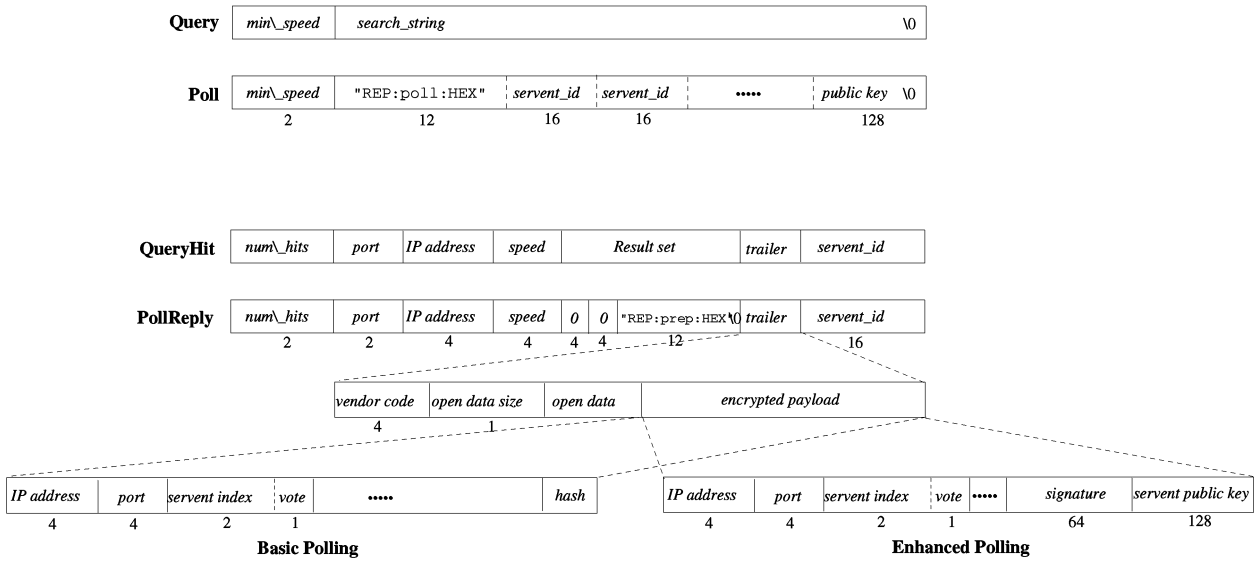

Fig. 5. A description of `P2PRep` messages.

string `REP:prep:HEX` is specified as file name; this triple is needed to identify the message. We then use the *private data* field in the *trailer* (introduced by version v1.3.0 of the `BearShare` servent) to store the encrypted payload. The payload, encrypted with RSA (1024), begins with the servent's *IP address* and *port*. A sequence of (*servent index*, *vote*) pairs follows, where *servent index* is a 2-byte field specifying the position (first, second, ...) of the *servent_id* in the corresponding `Poll` message, and *vote* is a 1-byte field denoting the vote. We note that, in the enhanced polling protocol, the encrypted payload must end with a 64-byte long *signature* for the whole message. Also, the *servent's public key* is appended.

5.2 The Architecture

Most Gnutella servents share the architectural pattern illustrated in the top portion of the diagram in Fig. 6 (we have omitted a few aspects, like `Push`, `Ping`, and `Pong` messages, that are not crucial and would obfuscate the symmetry of the architecture). We assume that the system has been designed with an object oriented model; inheritance can be used, for instance, to model `Poll` messages as specializations of `Query` messages, and `PollReply` messages as specializations of `QueryHit` messages. The general structure of the architecture is the following. Three components are directly connected to the network: the `Connection Manager`, the `Router`, and the `Poll Manager`. Protocol messages are illustrated as oriented arcs from their creator to their consumer and are distinguished by a black point at their origin (e.g., `QueryHit` messages are created by component `Query Processor` and consumed by the `Router`). The user interface permits creating queries (each managed by a `Query Manager`), to start polls on servent reputation (each managed by a `Poll Manager`), and to access `Shared resources` and reputations. We now analyze each of the architecture components, starting with the components that characterize servents that do not support `P2PRep`.

- The `Router` is a software component dedicated to message routing. An instance of the router is responsible for each direct open connection. Each `Router` is managed by a thread that waits for the arrival of messages on its connection or for requests generated by the servent. Upon reception of a message from the network connection, each `Router` checks if the message has already been received by the node and, if so, stops its propagation. The `Router` then checks received messages and discards those that are not compliant with the Gnutella protocol; some clients also police the network and do not propagate messages that have high values for parameters *Hops* and TTL, fearing that they would derive from opportunistic behavior from a node desiring to increase its horizon at the expense of global bandwidth.
- The `Connection Manager` manages incoming requests for the creation of a new connection (if successful, a new `Router` will be associated with the connection) or for the download of a resource available on the servent (using the HTTP protocol).
- When a user decides to make a search, a `Query Manager` instance is created which prepares a `Query` message and asks all the active routers in the node to insert it into the network.

After the `Router` has checked the correctness of received messages, it analyzes the message type and manages it appropriately, passing it to the appropriate processor. In particular, the `Router` is responsible to manage the `Query` and `QueryHit` messages, which it deals with as follows:

- `Query` messages are transferred to the other routers for their forwarding on the network (if their TTL is greater than zero) and are then delivered to a `Query Processor` that verifies the presence of the requested file in the local repository of shared files (`Shared resources`).

  If the `Query Processor` finds a match in the `Shared resources`, it creates a `QueryHit` message that is passed to the `Router` that returned the

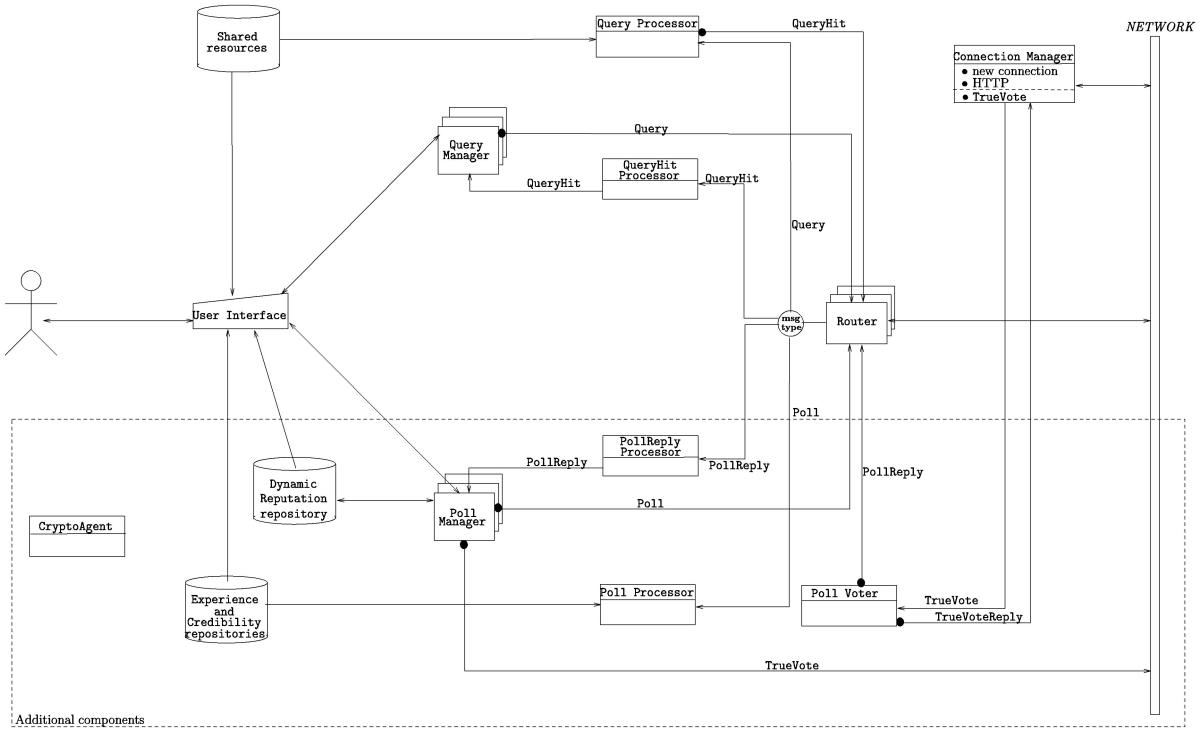

Fig. 6. Gnutella's information flow with protocol extensions.

`Query`. The `QueryHit` contains the location of the requested resource.

- `QueryHit` messages are routed following back the path taken by the corresponding `Query` message, using the routing tables that are also used to identify duplicates. When a message reaches the node that produced the query, the message is returned to the `Query Manager`, which usually adds its contents to a list of results and presents them on the search interface.

Our protocol requires extending this architecture with a few additional components, grouped in the dashed box in the lower half of Fig. 6. The additions can be summarized as follows:

- In a way similar to the management of queries, when a user expresses interest in acquiring reputation for a servent (or a list of servents), a `Poll Manager` instance is created. The `Poll Manager` immediately produces a `Poll` message and asks all active routers to propagate it through the P2P network.
- `Poll` messages are passed by the `Router` to a `Poll Processor` that verifies if there is any information on the servent in the local experience_repository. If the search is successful, an instance of `Poll Voter` is created that casts its vote within a new `PollReply` message that is sent back to the router.
- `PollReply` messages are routed back like `QueryHit` messages and reach the correct `Poll Manager`. The results of the poll are then used to populate a `Dynamic Reputation repository` that caches the poll results that have been obtained from the network. The `Poll Manager` is also responsible for checking vote authenticity and for creating `TrueVote` messages that are sent directly to the voting nodes.
- The `Connection Manager` is extended to process `TrueVote` requests, which are passed to the corresponding `Poll Voter`, which has to stay alive until a timeout period has passed.
- A `CryptoAgent` component offers the set of encryption functions required by `P2PRep`. `P2PRep` requires only standard encryption facilities: All that is needed is a public/private key pairs generation scheme, an encryption function, and a digital signature. For ease of implementation, we have chosen to use the most popular schemes providing the desired functionality.

Important modifications have to occur also in the `User Interface`, that has to offer a synthetic and effective representation of servent reputations. The `User Interface` is also the component where reputations are created. Typically, reputations are built monitoring every deletion of a resource, requested directly within the interface or detected at startup by a comparison of the currently offered resources with a list stored on a file. For each deletion, the user is asked if a negative vote must be expressed for the resource. The expressed vote contributes to the reputation of the servent(s) from which the resource had been downloaded.

6 Practical Considerations and Discussion

We describe here a few additional aspects that may clarify the potential of our solution and its possible integration with current P2P technologies.

Limited added cost. The implementation of the P2PRep protocol requires a certain amount of resources, in terms of both storage capacity and bandwidth, but this cost is limited and justified in most situations. The amount of storage capacity is proportional to the number of servents with which the servent has interacted. For the basic protocol, this will require adding at most a few bytes to the experience_expository, for an exchange that may have required the local storage of a file with a size of several millions of bytes. The enhanced version is more expensive in terms of local storage, but usually, the limiting resource in P2P networks is network bandwidth rather than storage.

The P2PRep protocol increases the traffic of the P2P network. The additional traffic can be distinguished in direct exchanges and broadcast requests. Direct exchanges require a limited number of short messages. For instance, direct connections are used to implement the TrueVote and AreYou messages. This exchange is very quick, as the messages contain only a few bytes and are directed only to some of the nodes that expressed their votes on same servent. Indeed, most performance models of P2P networks identify as the main limiting factor the aggregate bandwidth required by the exchange of broadcast messages. We evaluate the size of the messages required to estimate the reputation of servents offering a resource. First, we observe that many servents can be polled with a single Poll message. The size of a Poll message is proportional to the number of servents to enquire. As most implementations drop messages bigger than 64Kb [31], a Poll message can carry up to around 4,000 different *servent_id*. We assume that it is not necessary to ask on the reputation of that many servents. Instead, when a resource is offered by many servents, the client selects a subset of the servents to enquire about. Overall, the most expensive operation is the polling, which operates in the same way as a search. We observe that our service approximately doubles the traffic in a Gnutella network.

Several other optimizations can reduce the impact of P2PRep on network performance. For instance, reputations are cached on the nodes. Servents that have already been voted as reliable following a search can keep the reputation for the remainder of the session. Reputations may be kept across sessions, further reducing the number of polling requests (at the expense of an increase in the storage requirements and a decrease in the responsiveness of the network to node misbehavior).

Distribution of servents and resources. An aspect that has an impact on the performance of P2PRep is the distribution of servents and resources in the Gnutella network. It is reasonable to expect that servents and resources will be distributed nonuniformly, with a few servents offering many resources and many servents offering few resources. 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.

We analyzed the traffic on the current Gnutella network. The goal was not an extensive performance study like the one in [29]. Rather, our experiments were focused on the determination of a few parameters that were not, to our knowledge, available in existing literature and that are critical to evaluate the behavior of our protocol in the current Gnutella architecture.

An open source Gnutella client was modified and all the QueryHit and Pong messages traveling along the network were logged. As previously discussed, 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 produced 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 wanted 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 is also cited in [29]). Thus, we used the QueryHit messages to evaluate resources' concentration. We logged half a million QueryHit records. Almost half of the $500,000$ records were discarded, as they represented duplicate responses (i.e., generated by the same servent for the same resource in response to different queries). We considered the name and the size of a resource to recognize resource identity; 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 results of this analysis are reported in Fig. 7, showing the number of resources present on each servent. Servents are ordered according to the number of resources they offer. The graph in Fig. 7 uses logarithmic scales for both 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. 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 success of our protocol, as many servents offer only few resources and would probably not have offered enough resources to get a reputation in the network. We argue instead that skewness is advantageous to the protocol. Indeed, while only a few servents offer many resources, such servents have a considerably higher probability of being well known to network participants and answer with greater frequency to queries. Even if we consider the restriction set on the population of voters by the presence of the horizon in Gnutella, a polling mechanism is an effective mechanism to evaluate servent reputation.

Overload avoidance. Even if polling does not introduce an overload in the P2P network, our reputation service presents a considerable risk of focusing transfer requests on the servents that have a good reputation, reducing the degree of network availability. A possible solution to this problem is to consider reputable nodes as the sources of file

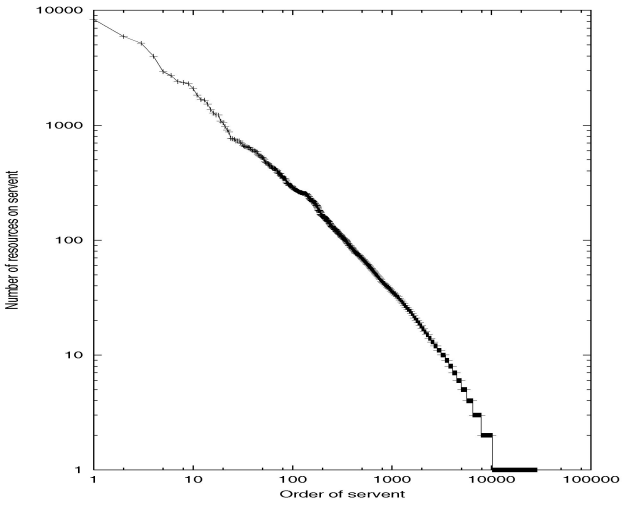

Fig. 7. Servent distribution: number of resources on servents.

identifiers of correct resources. The idea is to associate with every file a secure hash (e.g., using the SHA1 function), which is returned with the resource description. When a node identifies a resource it is interested in downloading, it first has to verify the offerers' reputation. As soon as a reputable offerer is identified, the requestor can interact directly with the offerer only to check the association between its servent_id and the `SHA1` signature. It can then request a download from any of the nodes that are exporting the resource with the same `SHA1` signature. Once the file transfer is completed, the correctness of the hash is checked.

Integration with intermediate P2P solutions. Intermediate P2P solutions (like FastTrack) identify nodes of the network characterized by an adequate amount of CPU power and network bandwidth, assigning to them the role of indexing what is offered on the network. The visible effect is a P2P network where response time and network congestion are greatly reduced, and users are not limited to searches on a portion of the resources offered on the network. In this situation, as in centralized solutions, when users connect to the network they are required to transfer to the indexing nodes a description of the resources they are sharing. For the implementation of our reputation mechanism, a synthesis of the votes on servents that each node has built and of its experience should also be transferred to the indexing node at the start of the session. A great opportunity in this context derives from a possible preprocessing, done on the indexing node, to associate a reputation with each servent. In this way, the reputation could be returned immediately in the result of a search. Since we have no access to a public description of this architecture, we did not consider this solution at the moment.

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. Digital certificates then have been introduced as a way to establish properties of a party, such as identity, accreditation, or authorizations (e.g., [6], [13], [15], [18], [22]). Based on these proposals, access control systems have also been enhanced, loosening the authentication prerequisite for authorization control and supporting credential-based authorizations where the ability of a peer to access resources depends on properties and certificates it can present, rather than on who it is, also allowing support for anonymous access and possibly introducing a form of negotiation (e.g., [7], [34]).

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 us 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 peers from distributing invalid or malicious content into the network are based on two main techniques: *micropayment* systems and *reputation*-based trust systems [24].

Micropayment approaches require peers to offer something of value in exchange of their participation in the system. In particular, peers wishing to export their contents should provide the counterpart with a payment, which may or may not have redeemable value. Redeemable payments can be used by the recipient as digital cash for acquiring services from other peers. Whether redeemable or not, micropayments can discourage malicious peers by forcing on them a cost before they can become an active part of the system. The idea is to balance what peers get from the network with what they provide, which also has the effect of preventing *free-rider* behaviors [3]. As an example, Mojo Nation (www.mojonation.net) is a distributed file sharing system based on a digital currency called *Mojo*. Peers earn Mojo when they provide resources (e.g., bandwidth and drive space) to the network. Mojo can then be used to request services from other peers. Therefore, to insert invalid content into the network malicious peers need first to provide an equal amount of resources.

Reputation models allow the expression and reasoning about *trust* in a peer based on its past behavior [27] and interactions other peers have experienced with it. 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 effectively used in electronic marketplaces as a measure of the reliability of participants [35]. 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 [9]) worked around the notion of *Web of trust* where trust relationships and reputations are managed by each participant which then distributes them. The common ancestor of such approaches is probably PGP (www.pgpi.org), that allows users to certify other user's public keys without need for a Certification Authority. Other approaches, 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/distrust it has in others on the basis of its own experience, and communicate this information to others. For instance, the proposal by Aberer and Desptovic [2] assumes that peers are usually "honest" and considers 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 enquire the network about existing complaints on their counterparts. Full support of negative reputations, however, can only be achieved at the price of sacrificing anonymity [17]. Also, support of negative reputations only is risky, as failure to retrieve existing complaints may result in trusting unreliable peers. As negative reputations can be nullified by taking on a fresh new identity, Free Haven proposes, like us, recording and exchanging positive reputations. However, the issues of maintaining and exchanging such reputations are not fully investigated. Other proposals on the same line distinguish reliability of peers depending on the specific context of interaction [1], [33].

8 Conclusions

We described a reputation management protocol for anonymous P2P environments that can be seen as an extension of generic services offered for the search of resources. The protocol is able to reconcile two aspects, anonymity and reputation, that are normally considered as conflicting. We demonstrated our solution on top of an existing Gnutella network.

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful comments. A preliminary version of this paper appeared under the title of *Choosing Reputable Servents in a P2P Network*, in the Proceedings of the 11th International World Wide Web Conference, Honolulu, Hawaii, 7-11 May, 2002. [11].

References

[1] A. Abdul-Rahman and S. Hailes, "Supporting Trust in Virtual Communities," *Proc. Hawaii Int'l Conf. System Sciences,* Jan. 2000.

[2] K. Aberer and Z. Despotovic, "Managing Trust in a Peer-2-Peer Information System," *Proc. 10th Int'l Conf. Information and Knowledge Management (CIKM 2001),* Nov. 2001.

[3] E. Adar and B. Huberman, "Free Riding on Gnutella," technical report, Xerox PARC, Aug. 2000.

[4] A. Bardossy, L. Duckstein, and I. Bogardi, "Combination of Fuzzy Numbers Representing Expert Opinions," *Fuzzy Sets and Systems,* vol. 57, pp. 173-181, 1993.

[5] S. Bellovin, "Security Aspects of Napster and Gnutella," *Proc. USENIX 2001,* June 2001.

[6] M. Blaze, J. Feigenbaum, J. Ioannidis, and A.D. Keromytis, "The Role of Trust Management in Distributed Systems Security," *Secure Internet Programming: Issues in Distributed and Mobile Object Systems,* 1998.

[7] P. Bonatti and P. Samarati, "Regulating Service Access and Information Release on the Web," *Proc. Seventh ACM Conf. Computer and Comm. Security,* 2000.

[8] S.J. Brams, "The Ams Nomination Procedure is Vulnerable to Truncation of Preferences," *Notices of the Am. Math. Soc.,* vol. 29, pp. 136-138, 1982.

[9] R. Chen and W. Yeager, "Poblano—A Distributed Trust Model for Peer-to-Peer Networks," JXTA Security Project White Paper, 2001.

[10] I. Clarke, O. Sandberg, B. Wiley, and T. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Proc. ICSI Workshop Design Issues in Anonymity and Unobservability,* July 2000.

[11] F. Cornelli, E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati, "Choosing Reputable Servents in a P2P Network," *Proc. 11th Int'l World Wide Web Conf.,* May 2002.

[12] R. Dingledine, M.J. Freedman, and D. Molnar, "The Free Haven Project: Distributed Anonymous Storage Service," *Proc. Workshop Design Issues in Anonymity and Unobservability,* July 2000.

[13] V. Doshi, A. Fayad, S. Jajodia, and R. MacLean, "Using Attribute Certificates with Mobile Policies in Electronic Commerce Applications," *Proc. 16th Ann. Computer Security Applications Conf. (ACSAC '00),* pp. 298-307, 2000.

[14] P. Druschel and A. Rowstron, "Past: A Large-Scale Persistent Peer-to-Peer Storage Utility," *Proc. Eight IEEE Workshop Hot Topics in Operating Systems (HotOS-VIII),* May 2001.

[15] C. Ellison SPKI certificate documentation, http://www.pobox.com/~cme/html/spki.html, 2002.

[16] R. Fagin, "Combining Fuzzy Information from Multiple Systems," *Proc. 15th ACM SIGACT-SIGMOD-SIGAR Symp. Principles of Database Systems,* June 1996.

[17] E.J. Friedman, P. Resnick, "The Social Cost of Cheap Pseudonyms," *J. Economics and Management Strategy,* vol. 10, no. 2, pp. 173-199, 2001.

[18] B. Gladman, C. Ellison, and N. Bohm, "Digital Signatures, Certificates and Electronic Commerce," http://citeseer.nj.nec.com/277887.html, 1999.

[19] L. Gong, "JXTA: A Network Programming Environment," *IEEE Internet Computing,* vol. 5, no. 3, pp. 88-95, May/June 2001.

[20] IP to Latitude/Longitude Server, Univ. of Illinois, http://cello.cs.uiuc.edu/cgi-bin/slamm/ip2ll.

[21] "Web Security—A Matter of Trust," *The World Wide Web J. (Special Issue),* R. Khare, ed., vol. 2, summer 1997.

[22] U. Maurer, "Modeling a Public Key Infrastructure," *Proc. Fourth European Symp. Research in Security and Privacy,* pp. 325-350, Sept. 1996.

[23] D. Moore, "Where in the World is Netgeo.Caida.Org?" *Proc. INET 2000,* June 2000.

[24] *Peer-to-Peer: Harnessing the Power of Disruptive Technologies,* A. Oram, ed. O'Reilly & Associates, Mar. 2001.

[25] V. Padmanabhan and L. Subramanian, "An Investigation of Geographic Mapping Techniques for Internet Hosts," *Proc. ACM-SIGCOMM '01,* Aug. 2001.

[26] M. Parameswaran, A. Susarla, and A.B. Whinston, "P2P Networking: An Information-Sharing Alternative," *Computer,* vol. 34, no. 7, pp. 31-38, July 2001.

[27] P. Resnick, R. Zeckhauser, E. Friedman, and K. Kuwabara, "Reputation Systems," *Comm. ACM,* vol. 43, no. 12, pp. 45-48, Dec. 2000.

[28] M. Ripeanu, "Peer-to-Peer Architecture Case Study: Gnutella Network," Technical Report TR-2001-26, Univ. of Chicago, Dept. of Computer Science, July 2001.

[29] S. Saroiu, P.K. Gummadi, and S.D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. Multimedia Computing and Networking,* Jan. 2002.

[30] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. 2001 Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm.,* 2001.

[31] S. Thadani, "Free Riding on Gnutella," technical report, LimeWire LLC, 2001, http://www.limewire.org.
[32] *The Gnutella Protocol Specification v0.4 (Document Revision 1.2).* June 2001, http://www.clip2.com/GnutellaProtocol04.pdf.
[33] B. Yu and M.P. Singh, "A Social Mechanism for Reputation Management in Electronic Communities," *Proc. Fourth Int'l Workshop Cooperative Information Agents (CIA),* July 2000.
[34] T. Yu, M. Winslett, and K. Seamons, "Interoperable Strategies in Automated Trust Negotiation," *Proc. Eighth ACM Computer and Comm. Security,* Nov. 2001.
[35] G. Zacharia, A. Moukas, and P. Maes, "Collaborative Reputation Mechanisms in Electronic Marketplaces," *Proc. 32nd Hawaii Int'l Conf. System Sciences,* Jan. 1999.

Ernesto Damiani received the laurea degree in ingegneria elettronica from Università di Pavia and the PhD degree in computer science from the Università di Milano. He is currently a professor in the Department of Information Technology at the University of Milan. His research interests include distributed and object-oriented systems, semistructured information processing, and soft computing. He is the vice-chair of the ACM Special Interest Group on Applied Computing (SIGAPP). He is the author, together with I. Sethi and R. Khosla, of the book *Multimedia MultiAgent Systems* (Kluwer 2001). He is a member of the IEEE.

Sabrina De Capitani di Vimercati received the laurea and PhD degrees, both in computer science, from the University of Milan in 1996 and 2001, respectively. She is an associate professor in the Department of Information Technology at the University of Milan. Her research interests are in the areas of information security, databases, and information systems. She has been an international fellow in the Computer Science Laboratory at SRI in California. She is corecipient of the ACM-PODS'99 Best Newcomer Paper Award. The URL for her web page is http://www.dti.unimi.it/~decapita. She is a member of the IEEE Computer Society.

Stefano Paraboschi received the laurea degree in ingegneria elettronica in 1990 and the PhD degree in ingegneria informatica in 1994, both from Politecnico di Milano. He is a professor in the Department of Engineering at the University of Bergamo. His main research interests are in the areas of databases, Web technologies, and security. The main topics he worked on are active databases, data warehouses, data-intensive Web sites, query languages for XML, access control for XML, and security for P2P applications. He is a coauthor of the book *Database Systems: Concepts, Languages and Architectures* (McGraw-Hill, 1999).

Pierangela Samarati is a professor in the Department of Information Technology at the University of Milan. Her main research interests are in data and application security, information system security, access control policies, models and systems, and information protection in general. She has been a computer scientist in the Computer Science Laboratory at SRI in California. She has been a visiting researcher in the Computer Science Department at Stanford University, California, and in the ISSE Department at George Mason University, Virginia. She is coauthor of the book *Database Security* (Addison-Wesley, 1995). She is corecipient of the ACM-PODS'99 Best Newcomer Paper Award. The URL for her web page is http://seclab.crema.unimi.it/~samarati. She is a member of the IEEE Computer Society.