Choosing Reputable Servents in a P2P Network

Fabrizio Cornelli
Dipartimento di Tecnologie dell'Informazione
Università di Milano
26013 Crema - Italy
fcornelli@crema.unimi.it

Ernesto Damiani
Dipartimento di Tecnologie dell'Informazione
Università di Milano
26013 Crema - Italy
damiani@dti.unimi.it

Sabrina De Capitani di Vimercati
Dipartimento di Elettronica
Università di Brescia
25123 Brescia - Italy
decapita@ing.unibs.it

Stefano Paraboschi
Dipartimento di Elettronica e Informazione
Politecnico di Milano
20133 Milano - Italy
parabosc@elet.polimi.it

Pierangela Samarati
Dipartimento di Tecnologie dell'Informazione
Università di Milano
26013 Crema - Italy
samarati@dti.unimi.it

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 nicely complements the 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.

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

P2P network, reputation, credibility, polling protocol

WWW2002, May 7–11, 2002, Honolulu, Hawaii, USA.
ACM 1-58113-449-5/02/0005.

1. INTRODUCTION

In the world of Internet technologies, peer-to-peer (P2P) solutions are currently receiving considerable interest [6]. P2P communication software is increasingly being used to allow individual hosts to anonymously share and distribute various types of information over the Internet [15]. While systems based on central indexes such as Napster [13] collapsed due to litigations over potential copyright infringements, the success of 'pure' P2P products like Gnutella [20] and Freenet [4] 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 [5, 9, 14, 17]. Still, there are several thorny issues surrounding research on P2P architectures [3].

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 copyright materials permitted by P2P systems. This opposition is testified by the recent lawsuit filed against P2P software distributors Grokster, KaZaA and MusicCity by the RIAA and MPAA, and by the previous successful lawsuit against Napster filed by the RIAA. 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.

Secondly, 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 [11], and allows malicious users to exploit the P2P infrastructure to freely distribute Trojan Horse and Virus programs. Some pratictioners 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 software undeniably increases the chances of being exposed, especially for home users who cannot rely on a security policy specifying which anti-virus 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 only 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.

Our digital reputations can be seen as the P2P counterparts of client-server digital certificates [7, 8], 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 re-acquiring 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. Secondly, 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. As we shall see, our technique can be easily integrated with existing P2P protocols.

2. ARCHITECTURES FOR PEER-TO-PEER NETWORKS

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. We focus on P2P networks for file exchange, where the P2P label clarifies that nodes have flexible roles and may function at the same time as clients and servers. Typically, P2P applications offer a default behavior: they immediately make available as servers all the files they retrieved as clients. For this dual nature of server and client, a node in a P2P network is called a *servent*.

The use of a P2P network for information exchange involves two phases. The first phase is the search of the servent where the requested information resides. The second phase, which occurs when a servent has identified another servent exporting a resource of interest, requires to establish a direct connection to transfer the resource from the exporting servent to the searching servent.

While the exchange phase is rather direct and its behavior is relatively constant across different architectures, the first phase is implemented in many different ways and it most characterizes the different solutions. We identify three main alternatives: *centralized indexes*, *pure P2P architectures*, and *intermediate solutions*.

The best representative of centralized solutions is Napster, a system dedicated to the exchange of audio files. Napster was the first P2P application to gain considerable success and recognition. It used a centralized indexing service that described what each servent of the network was offering to the other nodes. Based on its indexes, Napster was able to efficiently answer search queries originating from servents in the network and direct them to the servent offering the requested resource. Napster reached a peak of 1.5 million users connected at the same time, before being forced to activate filters on the content that users were offering, to eliminate from the indexes copyrighted materials. Combined with the introduction of a paid subscription mechanisms, this forced a rapid decline in the number of Napster users and currently the service is not operational. Other solutions were quick to emerge, to fill the void left by Napster, avoiding the centralization that permitted Napster to offer good performance, but also that forced it to take responsibility for the content that users were exchanging.

The best known representatives of pure P2P architectures are Gnutella and Freenet. Gnutella was originally designed by Nullsoft, owned by America OnLine, but was immediately abandoned by AOL and is currently maintained by a number of small software producers. Gnutella is a distributed architecture, where all the servents of the network establish a connection with a variable number of servents, creating a grid where each servent is responsible of transferring queries and their answers. Freenet is an open source architecture explicitly designed for the robust anonymous diffusion of information. Each resource is identified by a key, and support for searches is strictly based on this key. Freenet is designed to offer sophisticated services for the protection of the integrity and the automatic distribution of files near to the servents where requests are more frequent. Freenet is currently offering a low degree of usability, which limits its use to a relatively restricted number of adopters, compared with the other solutions.

Intermediate architectures have recently emerged. The best representative of this family is the product developed by FastTrack(`www.fasttrack.nu`), a company originally based in the Netherlands, and now owned by an Australian company. The FastTrack's software has been licensed to companies KaZaA, MusicCity, and Grokster. FastTrack distinguishes its servents in *supernodes* and nodes: supernodes are servents which are responsible for indexing the network content and in general have a major role in the organization of the network. A node is eligible to become a supernode

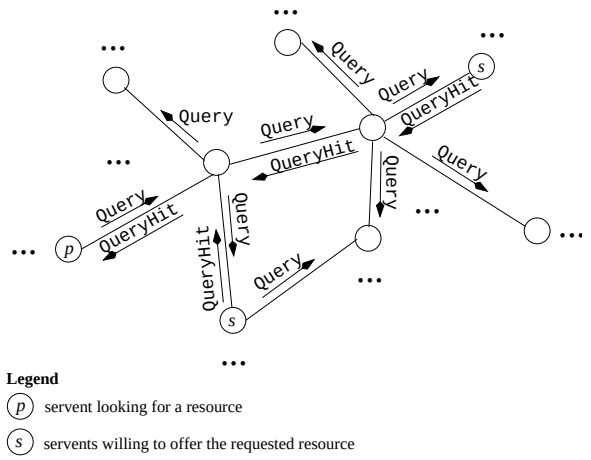

Figure 1: Locating resources in a Gnutella-like P2P environment

only if it is characterized by adequate resources, in terms of bandwidth and computational power. Files shared on the network are enriched with metadata, automatically generated or input by the user, that permit more precise searches. This solution is particularly successful: at the time of writing, February 2002, there are reports of 80 million downloads of the application, 1.5 million users connected on average at any time, and almost 2 billion files expected to be exchanged in the month.

We will use Gnutella throughout the paper as a reference, because it is an open protocol and simple open source implementations are available that permit to experiment with our protocol variants.

2.1 Basic description of Gnutella

Gnutella offers a fully peer-to-peer decentralized infrastructure for information sharing. 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, 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 TTL (Time To Live) that 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 forced to work on only a portion of the resources globally offered.

To search for a particular file, a servent p sends a broadcast `Query` message to every node linked directly to it (see Figure 1). The fact that the message is broadcast through the P2P network, implies that the node not directly connected with p will receive this message via intermediaries; they do not know the origin of the request. 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. 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`, inasmuch it requires the "pushed" servent to initiate the connection for downloading.

2.2 Security threats to Gnutella

Gnutella is a good testbed for our security provisions, as it is widely acknowledged that its current architecture provides an almost ideal environment for the spread of self-replicating malicious agents. This is due to two main features of Gnutella's design: *anonymous peer-to-peer communication* (searches are made by polling other Gnutella clients in the community and clients are anonymous as they are only identified by an opaque, self-appointed servent id), and *variety of the shared information* (the files authorized to be shared can include all media types, including executable and binary files). The former feature involves a weakness due to the combination of low accountability and trust of the individual servents. In an ordinary Internet-based transaction, if malicious content is discovered on a server the administrator can be notified. On Napster, if a user was caught distributing malicious content, his account could be disabled. In Gnutella, anyone can attach to the network and provide malicious content tailored to specific search requests with relatively small chance of detection; even blacklisting hostile IPs is not a satisfactory countermeasure, as there are currently no mechanisms to propagate this information to the network servents (in a pure distributed architecture there is no central authority to trust), and in many situations servents use dynamically assigned IPs. Gnutella clients are then more easily compromised than Napster clients and other file sharing tools. For instance, the well-known `VBS.Gnutella` worm (often mis-called the *Gnutella virus*) spreads by making a copy of itself in the Gnutella program directory; then, it modifies the `Gnutella.ini` file to allow sharing of `.vbs` files in the Gnutella program folder. Other attacks that have been observed rely on the anonymity of users: under the shield of anonymity, malicious users can answer to virtually any query providing tampered with information. As we shall see in Section 5, these attacks can be prevented or their effects relieved, increasing the amount of accountability of Gnutella servents.

3. SKETCH OF THE APPROACH

Each servent has associated 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. 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] Our approach encourages per-

[1]It must be noted that, while not compromising anonymity,

sistence as the only way to maintain history of a servent_id across transactions.

As illustrated in the previous section, in a Gnutella-like environment, a servent p looking for a resource broadcasts a query message, and selects, among the servents responding to it (which we call *offerers*), the one 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.

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 basic idea is as follows. 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 message requesting their opinion about the selected servents. All peers can respond 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. We present two flavors of our approach. 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 in weighting the votes received (p can judge some voters as being more credible than others).

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 quality of the poll means ensuring the integrity of each single vote (e.g., detecting modifications to votes in transit) and rule out the possibility of dummy votes expressed by servents acting as a clique under the control of a single malicious party. In the next section we describe how these issues are addressed in our protocols.

4. REPUTATION-BASED SOURCE SELECTION PROTOCOLS

Both 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 [2] 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 $(\text{PK}_i,\text{SK}_i)$ 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 $\{M\}_K$ and $[M]_K$ to denote the encryption and signature, respectively, of a message M under key K. Also, in illustrating the protocols, we will use p to denote the protocol's initiator, $\mathcal{S}$ to denote the set of servents connected to the P2P network at the time p sends the query, O to denote the subset of $\mathcal{S}$ responding to the query (*offerers*), and V to denote the subset of $\mathcal{S}$ responding to p's polling (*voters*). A message transmission from servent x to servent y via the P2P network will be represented as $x \longrightarrow y$, where "$*$" appears instead of y in the case of a broadcast transmission. A direct message transmission (outside the P2P network) from servent x to servent y will be represented as $x \xrightarrow{\text{D}} y$.

persistent identifiers introduce linkability, meaning transactions coming from a same servent can be related to each other.

4.1 Basic polling

The basic polling solution, illustrated in Figure 2, works as follows. Like in the conventional Gnutella protocol, the servent p looking for a resource sends 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 ⟨IP,port⟩, which p can use for downloading. Then, p selects its top list of 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 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 an opinion on any of the servents in the list, send back a `PollReply` expressing their votes and declaring their ⟨IP,port⟩ pair (like when responding to queries). The poll reply is encrypted with the public key provided by p to ensure its confidentiality (of both the vote and the voters) when in transit and to allow p to check its integrity. Therefore, as a consequence of the poll, 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 decryption to detect tampered with votes and discards them. Second, p detects votes that appear suspicious, for example since they are coming from IPs suspected of representing a clique (we will elaborate more on this in Section 4.4). Third, p selects a set of voters that it directly contacts (by using the ⟨IP,port⟩ pair they provided) to check whether they actually expressed that vote. For each selected voter v_j, p directly sends a `TrueVote` request reporting the votes it has received from v_j, and expects back a confirmation message `TrueVoteReply` from v_j confirming the validity of the vote. This forces potential malicious servents to pay the cost of using real 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 p's poll public key, is not visible to them. Upon assessing correctness of the votes received, p can finally select the offerer it judges as its best choice. Different criteria can be adopted and any servent can use its own. For instance, p can choose the offerer with the highest number of positive votes,

[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.

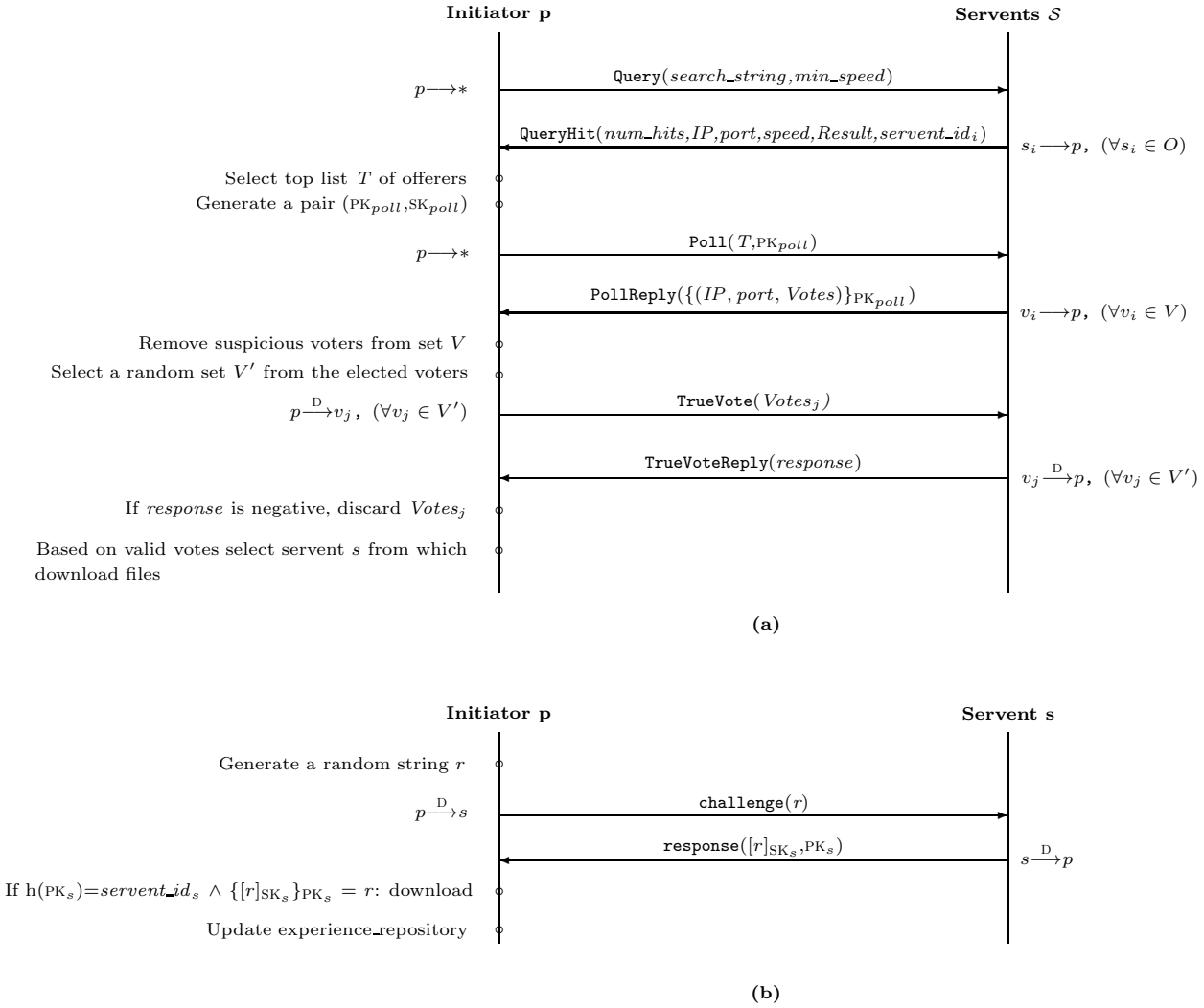

Figure 2: Sequence of messages and operations in the basic polling protocol (a) and download of files from the selected servent (b)

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.

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 that s has declared, then p will know that it is actually talking to s. Note that the challenge-response exchange is done via direct communication, like the download, in order 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 initiate the download and, depending on its satisfaction for the operation, update its reputation information for s.

4.2 Enhanced polling protocol

The enhanced polling protocol differs from the basic solution by requesting voters to provide their servent_id. Intuitively, while in the previous approach 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 weight the votes they express when responding to a polling request. The approach, illustrated in Figure 3, works as follows. Like for the basic case, 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 the public key PK_{poll}, the public key PK_i of the voter and its vote declarations signed with the corresponding private key SK_i. The vote declaration contains the pair ⟨IP,port⟩ 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

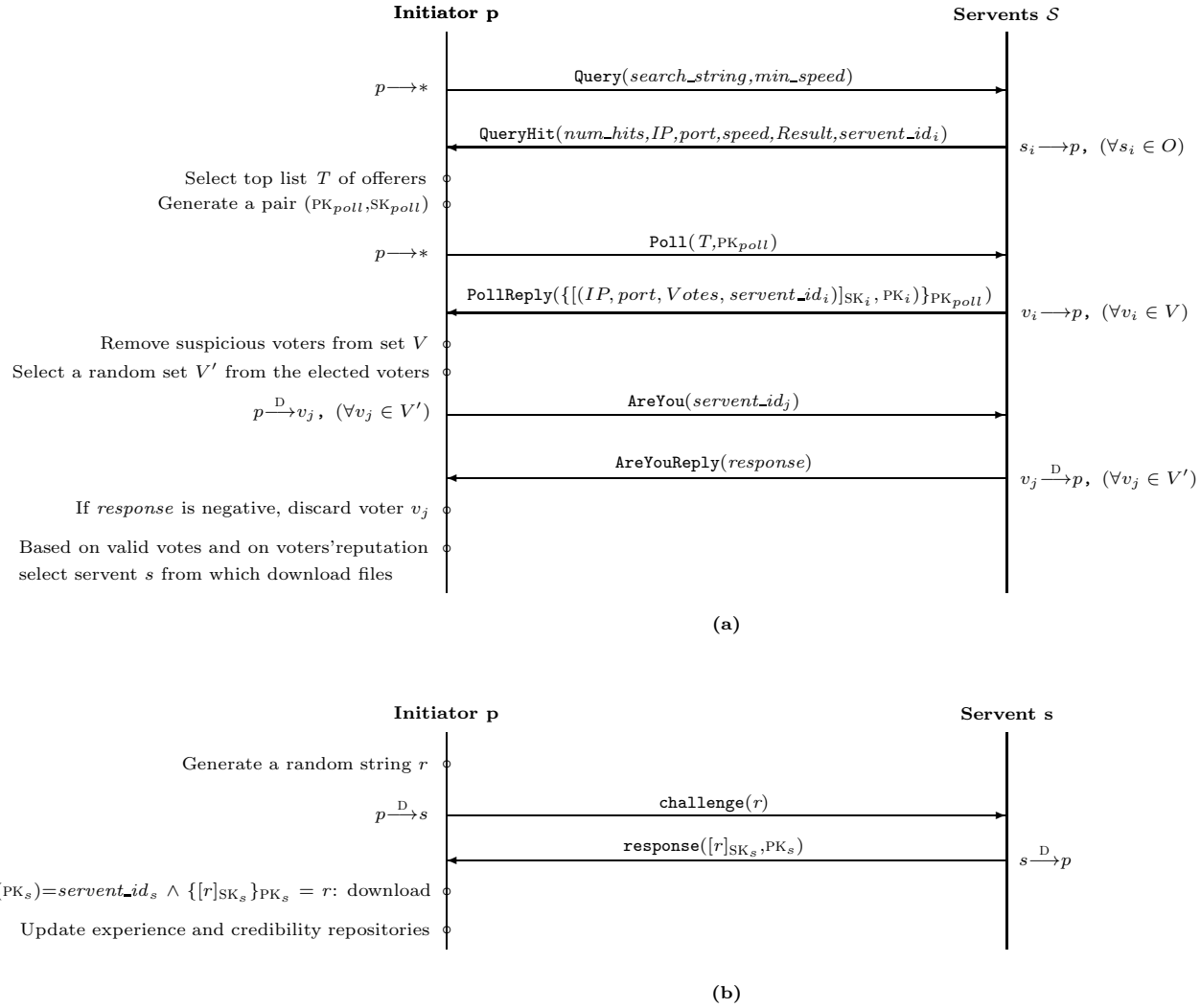

Figure 3: Sequence of messages and operations in the enhanced polling protocol (a) and interactions with the selected servent (b)

confidentiality and 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: they could have been expressed only by a party knowing the servent_id private key. Again, after collecting all the replies to the poll, p carries out an analysis of the votes received removing suspicious votes and then selects a set of 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 anymore to check the integrity of the vote as the vote's signature guarantees it). Selected voters are then directly contacted, via the ⟨IP,port⟩ pair they provided with a message `AreYou` reporting the servent_id that was associated with this pair in the vote.[3] Upon this direct contact, the voter responds with a `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 according to the *i)* connection speed, *ii)* its own reputation about the servents, and *iii)* the reputations expressed in the votes received. While in the basic polling all votes were considered equal (provided removal of suspicious votes or aggregation of suspected cliques), the knowledge about the servent_ids of the voters allows p to weight 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 Subsection 4.3).

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 whose public key PK_s's digest corresponds to 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.

4.3 Maintaining servents' reputations and credibilities

[3]Note that it is not sufficient to determine that the ⟨IP,port⟩ is alive since any servent_id could abuse of ⟨IP,port⟩ pairs available but disconnected from the P2P network.

When illustrating the protocols we simply 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 express (*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. Here, we illustrate the approach we adopted in our current implementation.

4.3.1 Representing Reputations

Each servent s maintains an *experience_repository* as a set of triples (*servent_id*, *num_plus*, *num_minus*) associating with each servent_id the number of successful (*num_plus*) and unsuccessful (*num_minus*) downloads s experienced. Servent s can judge a download as unsuccessful, for example, if the downloaded resource was unreadable, corrupted or included malicious content. The experience_repository will be updated after each download by incrementing the suitable counter, according to the download outcome. Keeping two separate counters (for bad and good experiences) provides the most complete information.

4.3.2 Translating Local Reputations into Votes

The simplest form of vote is a binary value by which a vote can be either positive (`1`) or negative (`0`). Whether to express a positive or a negative opinion can be based on different criteria that each voter can independently adopt. For instance, a peer may decide to vote positively only for servents with which it never had bad experiences (*num_minus*=0), while others can adopt a more liberal attitude balancing bad and good experiences.

While we adopted simple binary votes, 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%). The only constraint for the approach to work properly is that the scale on which one expresses votes should be communicated to the poller.

4.3.3 Representing Credibilities

Each servent x maintains a *credibility_repository* as a set of triples (*servent_id*, *num_agree*, *num_disagree*) associating with each servent_id its accuracy in casting votes. Intuitively, *num_agree* represents the number of times the servent_id's opinion on another peer s (within a transaction in which s was then selected for downloading) matched the outcome of the download. Conversely, *num_disagree* represents the number of times the servent_id's opinion on another peer s (again, within a transaction in which s was then selected for downloading) did not match the outcome of the download. A simple approach to the credibility_repository maintenance is as follows. At the end of a successful transaction, the initiator p will increase by one the *num_agree* counter of all those servents that had voted in favor of the selected servent s and will increase by one the *num_disagree* counter of all those servents that had voted against s. The vice versa happens for unsuccessful transactions.

4.4 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 voters. 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. 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 to generate additional network traffic. An alternative, more robust, approach currently used by many tools, such as IP2LL [10] and NetGeo [12], computes IP clustering by accessing a local `Whois` database to obtain the IP *address block* that includes a given IP address.[4] 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 U.S., and the IP addresses of such cluster all belong to a single address block [16]. 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. Then, we use resulting *cluster votes* to obtain the final poll outcome, computed as a weighted average of the cluster's votes, where weights are inversely related to cluster sizes. 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.

5. 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,

[4]In our current prototype, a local `Whois` query is generated for all the IP addresses in the voters' set, and query results are used for computing the distinct address blocks that include the voters' IP addresses.

thus contributing to a cooperative increase of the P2P community's ethics. Secondly, 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 [18] 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 4 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 as it likes. Credentials can be coupled with either of our polling processes: in the basic protocol case, the servent_ids of the direct voters remain anonymous while the ones of those that voted indirectly are disclosed. Finally, when a P2P system is used as a private infrastructure for information sharing (e.g., in corporate environments), P2PRep votes 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.

5.1 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 [3]. 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.

5.1.1 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 (like the Gnutella virus mentioned in Section 2.2). Currently, this attack is particularly common as it requires virtually no hacking of the software client. Both the simple and enhanced version of our protocol are aimed at solving the problem of impersonation attacks. When Alice discovers the potentially harmful content of the information she downloaded from David, she will update 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.

5.1.2 Man in the Middle

This kind of attacks 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:

1. Alice broadcasts a `Query` and Bob responds.
2. David intercepts the `QueryHit` from Bob and rewrites it with his IP address and port instead of Bob's.
3. Alice receives David's reply.
4. Alice chooses to download the content from David.
5. David downloads the original content from Bob, infects it and passes it on to Alice.

A variant of this attack relies on push-request interception:

1. Alice generates a `Query` and Carl responds.
2. Alice attempts to connect but Carl is firewalled, so she generates a `Push` message.
3. David intercepts the push request and forwards it with his IP address and port.
4. Carl connects to David and transfers his content.
5. 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 just before downloading. In order to impersonate Bob (or Carl) in this phase, David should know Bob's private key and be able to design a public key whose digest is Bob's identifier. Therefore, both versions of this attack are successfully prevented by our protocols.

6. IMPLEMENTING P2PREP IN THE GNUTELLA ENVIRONMENT

We are nearing completion of an implementation of our protocol as an extension to an existing Gnutella system. In this section, we describe how the P2PRep protocol is implemented and the modification it requires to a standard Gnutella servent's architecture.

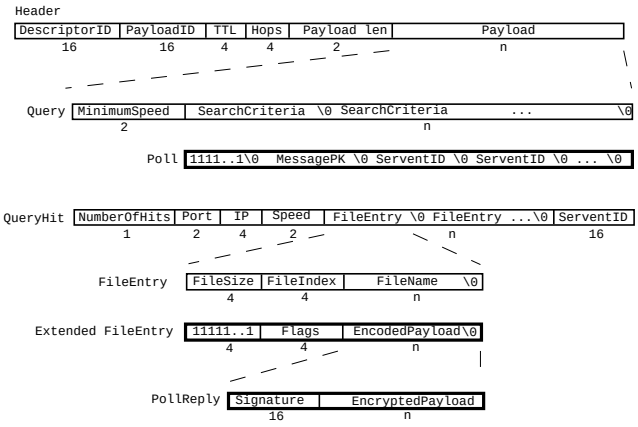

Figure 4: A description of P2PRep messages

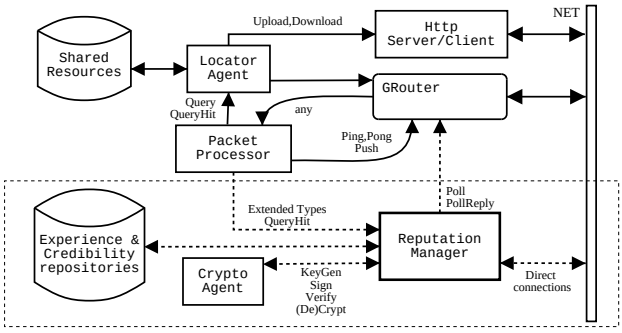

Figure 5: Gnutella's Information Flow with protocol extensions

6.1 P2PRep messages

To keep the impact of our proposed extension to a minimum, we use a *piggyback* technique: all P2PRep messages are carried as payload inside ordinary `Query` and `QueryHit` messages. P2PRep messages are summarized in Figure 4, which also shows their structure. Specifically, P2PRep encapsulation relies on the field `SearchCriteria` (a set of null-terminated strings) in the `Query` message and on the `FileEntry` fields of `QueryHit`. By carefully choosing message encoding, P2PRep broadcast messages (e.g., `Poll`) are stored in the `SearchCriteria` field of a `Query` message and will be understood by all P2PRep-compliant servents, while others will consider them as requests of (unlikely) filenames and simply ignore them. In turn, the `QueryHit` standard message is composed of `NumberOfHits` elements of a `FileEntry` containing a set of triples `(FileSize, FileIndex, FileName)`. We use these triples for encoding P2PRep unicast messages (e.g., `PollReply`) sent as replies to previous broadcasts. In order to ensure that our piggybacked messages are easily distinguished from standard `QueryHit`s and, at the same time, that they are safely ignored by standard Gnutella servents, P2PRep unicasts are encoded into the `Filename` field, while the `FileIndex` and `FileSize` fields specify the type of the message and the encoding of the payload. The internal structure of P2PRep messages is very simple: `Poll` is an anonymous broadcast message that contains a servent_id (or a set of them) and a session public key which is generated for each poll session. When a servent needs to poll the net about a peer, it generates a temporary key pair, and sends the poll public key with the `Poll` message itself. The `PollReply` message is encrypted and signed by the sender, with a persistent servent key. In our current design, the actual structure of the `PollReply` message depends on a parametric encoding function, stored in the `FileIndex` field of the `QueryHit` carrier. However, all `PollReply` messages contain an `EncryptedPayload` composed of a set of encrypted strings. Each of these strings holds a ⟨`ServentID, PollValue`⟩ pair.

6.2 The architecture

Although several implementations are available, most Gnutella servents share a common architectural pattern, that can be better understood looking at the information flow represented in Figure 5. In a standard architecture, two components are directly connected to the net: the `http` server/client, used for uploads and downloads, and the `GRouter`, a software component dedicated to message routing. This component carries messages from the net to the `Packet Processor`, a switch able to unpack messages, identify their type and deliver them to the right manager component. For instance, `Query` messages are delivered to a `Locator Agent` that verifies the presence of the requested file in the local repository of shared files (`Shared Resources`). All the other messages are rerouted on the net. If the `Locator Agent` finds a match in the `Shared Resources`, the Gnutella servent sends a `QueryHit` message through the `GRouter` specifying its location to the requestor. Our protocol requires complementing this architecture with three additional components (enclosed by a dotted line in Figure 5). The `Reputation Manager` is notified when query hits occur and receives all `Query` and `QueryHit` messages carrying P2PRep extensions. Those messages are processed in order to choose the best servent based on the reputation and credibility data stored in the `Experience` and `Credibility` repositories. In order to assess peers' reputations, the `Reputation Manager` sends and receives `Poll` and `PollReply` messages via the `GRouter`, as well as service messages for key handling (not shown in Figure 5). The `Reputation Manager` is linked to a `CryptoAgent` component encapsulating the set of encryption functions required by P2PRep. P2PRep requires only standard encryption facilities: all 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 functionalities. Namely, `RSA` for keys and `MD5` message digest of the payload for digital signatures.

7. COMMENTS 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 cost*: The implementation of our polling service 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 to add at most a few bytes to the experience_repository, 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 normally, the limiting resource in P2P networks is network bandwidth rather than storage. The most network intensive phase of our protocol is the polling phase, where a `Poll` request is broadcast to the network and `PollReply` responses are transmitted back by nodes participating in the poll. The checking phase may be also quite heavy on network bandwidth when the check has to analyze all the votes, but in normal situations, when votes are not forged, the random selection of a limited set of checks makes it a modest addition to the network load. In conclusion, the most expensive operation is the polling phase, which operates in the same way as a search. We can then assume that our service would approximately double the traffic in a Gnutella network.

- *Concentration of servents*: as we have already observed, the Gnutella protocol limits the portion of the network that each node can see. This means that servents will have a high probability of exhibiting a sufficient number of votes supporting their reputation in the portion of the network that a node in a particular instant sees only if they have a considerably greater number of votes globally. We do not consider this a strong limitation of the approach. As some studies have indicated [1, 19], current P2P solutions show a clear distinction between participants to the network, with a relatively small portion of servents offering a great number of resources, and a great number of servents (*free riders*) which do not share resources but only exploit what is offered by other participants. In this situation, it should be possible to identify, even in small portions of the network, servents that will exhibit an adequate 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 identifiers of correct resources. The idea is to associate with every file an `MD5` signature, 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 `MD5` signature. It can then request a download from any of the nodes that are exporting the resource with the same `MD5` signature. Once the file transfer is completed, the signature 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 immediately transfer to the indexing nodes the description of the resources they are sharing. For the implementation of our reputation mechanism, the votes on servents that each node has built and 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.

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. This paper represents a first step towards the development of a self-regulating system for preventing malicious behavior on P2P networks.

9. ACKNOWLEDGMENTS

The work reported in this paper was partially supported by the Italian MURST DATA-X project and by the European Community within the Fifth (EC) Framework Programme under contract IST-1999-11791 – FASTER project.

10. REFERENCES

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

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

[3] S. Bellovin. Security aspects of Napster and Gnutella. In *Proc. of USENIX 2001*, Boston, June 2001.

[4] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, July 2000.

[5] 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.

[6] P. Druschel and A. Rowstron. Past: A large-scale persistent peer-to-peer storage utility. In *Proc. of the Eighth IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Schoss Elmau, Germany, May 2001.

[7] Carl Ellison. SPKI certificate documentation. http://www.pobox.com/∼cme/html/spki.html.

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

[9] L. Gong. JXTA: A network programming environment. *IEEE Internet Computing*, 5(3):88–95, May/June 2001.

[10] IP to latitude/longitude server. University of Illinois. http://cello.cs.uiuc.edu/cgi-bin/slamm/ip2ll.

[11] R. Khare, editor. *Web Security – A Matter of Trust*, volume 2. The World Wide Web Journal (Special Issue), summer 1997.

[12] D. Moore. Where in the world is netgeo.caida.org? In *Proc. of INET 2000*, Stockholm, Sweden, June 2000.

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

[14] Openprivacy. http://www.openprivacy.org.

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

[16] V. Padmanabhan and L. Subramanian. An investigation of geographic mapping techniques for internet hosts. In *Proc. of ACM-SIGCOMM'01*, San Diego, CA, US, August 2001.

[17] M. Parameswaran, A. Susarla, and A.B. Whinston. P2P networking: An information-sharing alternative. *IEEE Computer*, 34(7):31–38, July 2001.

[18] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. Technical Report TR-2001-26, University of Chicago, Department of Computer Science, July 2001.

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

[20] *The Gnutella Protocol Specification v0.4 (Document Revision 1.2)*, June 2001. http://www.clip2.com/GnutellaProtocol04.pdf.