Centralized Security Labels in Decentralized P2P Networks

Nathalie Tsybulnik, Kevin W. Hamlen and Bhavani Thuraisingham
Department of Computer Science
University of Texas at Dallas
Richardson TX 75080, USA
ntsybulnik@student.utdallas.edu, {hamlen, bhavani.thuraisingham}@utdallas.edu

Abstract

This paper describes the design of a peer-to-peer network that supports integrity and confidentiality labeling of shared data. A notion of data ownership privacy is also enforced, whereby peers can share data without revealing which data they own. Security labels are global but the implementation does not require a centralized label server. The network employs a reputation-based trust management system to assess and update data labels, and to store and retrieve labels safely in the presence of malicious peers. The security labeling scheme preserves the efficiency of network operations; lookup cost including label retrieval is $O(\log N)$, where N is the number of agents in the network.

1. Introduction

Ever since Napster [1] was introduced, peer-to-peer (P2P) systems have become a ubiquitous technology for data dissemination. Napster focused exclusively on music exchange, but later more general-purpose systems followed, including Gnutella [2], KaZaA [3], LimeWire [4], and many others. Such P2P systems have enjoyed great success in numerous application domains because they offer load balancing of computational resources, redundant storage, data permanence, and low-cost deployment.

However, existing P2P systems offer relatively few security guarantees to users. Data shared over these networks has low confidentiality because it might potentially be divulged to any peer, and it has low integrity because peers can lie about the content of the data they serve. For example, malicious peers can easily propagate (low-integrity) malicious code over today's P2P networks by publishing it under a misleading name. Recent studies [5, 6] have concluded that at as much as 68% of all executable content in KaZaA and 15% of all files exchanged over LimeWire contain malware.

These P2P network implementations also offer only weak privacy guarantees. Malicious peers can with low overhead generate a list of all data served by any given peer, and can generate a list of all peers that serve any particular item of data. These drawbacks make traditional P2P network implementations unsuitable for venues where privacy, data confidentiality, and data integrity are important to users.

In order to address these deficiencies, we have developed Penny, a P2P network in which data objects are augmented with reputations in the form of confidentiality and integrity labels. These reputations provide peers a basis for deciding whether or not to serve or download data. For example, data with a low integrity label is more likely to contain a virus or corrupted content, so peers might avoid downloading it. Dually, peers might refuse to serve data with a high confidentiality label to peers that they do not trust. To help peers evaluate the trustworthiness of other peers, Penny includes a reputation-based trust management system based on EigenTrust [7]. Penny also allows peers to withhold object ownership information from other peers in the network, allowing them to publish data anonymously.

Object and peer reputations are centralized in Penny such that each reputation is a function of the individual opinions of all peers. However, these centralized reputations are stored in a decentralized fashion such that the computational expense of tracking and communicating global reputations is spread roughly evenly across all peers in the network. In particular, for each object or peer, Penny assigns k peers to track its global reputation, where k is a constant chosen at network initialization. Peers cannot choose which reputations they are assigned to track, so with high probability at least $k/2$ of these peers report the reputation accurately, preventing malicious peers from effectively subverting the reputation.

We are currently implementing Penny client software in Java, and our early prototyping has significantly influenced the network design presented in this paper. Penny clients function like typical P2P clients except that the list of ob-

jects displayed to the user in response to a query is augmented with a global integrity and confidentiality label for each object, and the list of servers that offer each object is augmented with a global trust value for each server. Users can choose which object to download (if any) based on its integrity label, and can choose which server to download it from based on the server's trust value. They can also choose whether to serve an object to a particular requester based on the object's confidentiality label and the requester's trust value. Peers can provide feedback after each transaction, which influences the labels and trust values reported for future queries submitted to the network.

In the analysis of our system, we consider four classes of attacks:

- A malicious peer or collective might spread corrupt or incorrect data. For example, the malicious peer or collective might spread malicious code or circulate false facts.

- A malicious peer or collective might attach incorrect security labels to data. In particular, low-integrity data might be assigned a high-integrity label, or high-confidentiality data might be assigned a low-confidentiality label.

- A malicious peer or collective might attempt to learn which peers own certain data, perhaps as a prelude to staging additional attacks against those peers.

- A malicious peer or collective might attempt to generate a list of all data served by a particular peer, violating that peer's privacy.

We do not consider attacks upon the network overlay itself, such as message misrouting, message tampering, or denial of service attacks. These attacks are beyond the scope of this paper, but could be addressed with various techniques, such as digital signatures, delivery receipts, and non-deterministic routing. In addition, security labels in our system are treated as advice to peers rather than enforced security policies. That is, we do not enforce the requirement that high-trust peers never obtain low-integrity data, or that low-trust peers never obtain high-confidentiality data. Enforcing such policies using our labeling scheme is the subject of future work.

The organization of this paper is as follows. Related work is discussed in §2. An overview of Penny's design is provided in §3. In §4, the Penny algorithm is defined in detail, and we discuss security properties enforced by the design in §5. The paper is summarized with directions for future work in §6.

2. Related Work

P2PRep [8] implements integrity labels and reputation-based trust management for the Gnutella [2] P2P network. Integrity labels and trust values are acquired in P2PRep by polling a large number of peers using broadcast messages. Poll responses are then aggregated by the requesting peer to estimate the desired integrity label or trust value (along with trust values for all peers whose opinions were acquired by polling). This strategy has the advantage of being implementable atop the existing Gnutella network protocol, but it has the disadvantage that labels and trust values are not global and are not guaranteed to converge. That is, the integrity label or trust value obtained will depend on which agents were polled, which in turn depends upon the poller's placement within the P2P network. Two peers at different locations in the network might therefore consistently derive different reputations for the same resource. Broadcast messages can also be expensive, requiring $O(b^d)$ messages to be sent, where b is the branching factor of the network and d is a *time to live* parameter dictating the maximum depth of the tree of peers being polled.

Penny's algorithm for efficiently circulating data is based on the Chord algorithm [9]. Chord assigns an identifier to each peer, and arranges peers in a ring structure sorted by identifier. Each peer maintains a *finger table* of size m, where 2^m is the size of the identifier space. This enables peers to locate and contact the peer with a given identifier in $O(\log N)$ message hops, where N is the number of agents in the network. Each shared data object also has a single *key-holder* peer, who is charged with directing requesters of that object to peers that own it. To request an object, a peer can locate its key-holder in $O(\log N)$ message hops, whereupon the key-holder responds with a list of servers from which the object can be downloaded.

Alternatives to Chord include CAN [10], Pastry [11], and Tapestry [12]. These systems offer distributed, scalable, and efficient search systems for P2P networks; however, they do not include data security or privacy enforcement mechanisms. Our work extends Chord by providing a framework for maintaining centralized security labels for data shared over a Chord network.

Trust management systems are a useful tool for identifying warning signs for potential malicious behavior. Such systems typically assign trust levels to principals as various actions appear in the system. In this way they predict future behavior based on the past experiences of other users. There are three major types of trust management systems. *Reputation-based* systems use knowledge of a peer's reputation (gathered through personal or indirect experience) to determine the trustworthiness of another peer. Some examples of reputation-based trust management systems are EigenTrust [7], DMRep [13], P2PRep/XRep [8],

Sporas and Histos [14], PeerTrust [15], NICE [16], and DCRC/CORC [17]. In contrast, *policy-based* trust management systems, such as PolicyMaker [18], derive a trust level for each peer based on supplied credentials. Finally, trust management systems based on *social networks* determine trust by analyzing a complex social network. Of this type, Marsh [19] was one of the first to formally express trust from a sociological and psychological perspective. Some other examples of social network systems are Regret [20] and NodeRanking [21].

Penny incorporates a reputation-based trust management system based on EigenTrust [7]. EigenTrust is a secure, distributed trust management system that maintains a globalized trust value for each peer. These globalized trust values are obtained by an iterative computation that approximates the left eigenvector v of the matrix T of all local trust values in the network. That is, if we define element T_{ij} to be the degree to which peer a_i trusts peer a_j, then the left eigenvector v of matrix T measures each peer a's global trust based on how much each peer trusts a, how much each peer trusts the peers who trust a, etc.

To keep the algorithm scalable and robust, eigenvector v is computed in a distributed and redundant fashion, where k different peers are responsible for computing each element of v. Whenever peers a_i and a_j are involved in a transaction, they report feedback to the peers responsible for computing eigenvector elements v_i and v_j. These peers are referred to as *score-managers* for agents a_i and a_j because elements v_i and v_j are the global trust values of peers a_i and a_j, respectively. Score-managers for a given peer are determined via a family of k hash functions applied to the peer's identifier. Thus, acquiring a peer's trust value requires $O(k \log N)$ messages in an EigenTrust system built atop Chord. Penny improves upon this performance. In §3.2 we show that using the Penny protocol trust values can be retrieved using only $O(\log N + k)$ messages.

Penny peers desiring confidentiality must send some messages anonymously to other peers. Anonymizing tunnels are a powerful technology for accomplishing this within peer-to-peer networks. Penny therefore supports the Tarzan system [22, 23], which implements anonymizing tunnels for Chord networks. In Tarzan, a peer desiring anonymity routes its messages through a tunnel of randomly chosen peers. Multilayer encryption and randomly generated cover traffic are used to prevent any peer in the tunnel from learning whether its successor is the message originator or just another hop in the tunnel. Tarzan tunnels are bidirectional, allowing recipients of anonymous messages to reply without knowing the identity of the message originator. The approach has proved to be both flexible and scalable, requiring little overhead above that incurred by Chord's existing message-routing protocol.

3. System Overview

In this section, we provide a high-level overview of the structure of a Penny network, beginning with definitions of important concepts. Details of the algorithm are presented in §4.

3.1. Definitions

Agents. We refer to the peers in a P2P network as *agents*. Each agent a is assigned an *identifier* id_a by applying a one-way, deterministic hash function to its IP address and port number. We assume that identifiers are unique and that agents cannot influence which identifier they are assigned. An agent's identifier determines its position in the network's ring structure. When agents are arranged in a ring, each agent has a *predecessor* $pred(a)$ and a *successor* $succ(a)$. We refer to the interval $[id_a, id_{succ(a)} - 1]$ as the *identifier range* of agent a.

Objects and keys. An *object* o is an atomic item of data (e.g., a file) shared over a P2P network. Each object also has a unique identifier id_o (e.g., a file name). Objects can be owned by multiple agents. A single *key* is associated with each object and each agent. The keys for object o and agent a are defined by $key_o := h(id_o)$ and $key_a := h(id_a)$ respectively, where h is a one-way, deterministic hash function over the domain of identifiers.

Local confidentiality and integrity labels. Each object o is labeled with a measure of its integrity and confidentiality levels. We denote the integrity and confidentiality labels assigned to object o by agent a as $i_a(o)$ and $c_a(o)$, respectively. Integrity labels measure data quality; confidentiality labels measure who should be permitted to own the data. In Penny, *confidentiality* and *integrity labels* are modeled as real numbers from 0 to 1 inclusive, with 0 denoting lowest confidentiality and integrity and 1 denoting highest confidentiality and integrity.

Local trust values. Trust measures the belief that one agent has that another agent will behave as expected or promised. Each ordered pair of agents (a_1, a_2) has a *local trust value* denoted $t_{a_1}(a_2)$ that measures the degree to which agent a_1 trusts agent a_2. Like confidentiality and integrity labels, trust values range from 0 to 1 inclusive. EigenTrust [7] is an example of a trust management system that employs trust values normalized to this range.

Key-holders and score-managers. Each agent a_1 in the Penny network is assigned a (not necessarily unique) *key range*, denoted $kr(a_1)$. Agent a_1 is charged with tracking

the integrity and confidentiality labels assigned to all objects o that satisfy $key_o \in kr(a_1)$. In addition, agent a_1 tracks the trust values assigned to all agents a_2 satisfying $key_{a_2} \in kr(a_1)$. Whenever $key_o \in kr(a_1)$ holds, we refer to agent a_1 as a *key-holder* for object o, and we refer to object o as a *daughter object* of agent a_1. Likewise, whenever $key_{a_2} \in kr(a_1)$ holds we refer to a_1 as a *score-manager* for agent a_2, and we refer to agent a_2 as a *daughter agent* of agent a_1. Every peer in a Penny network acts as both a key-holder for some objects and a score-manager for some peers.

Global labels and trust values. Key-holders with a common key-range use the local integrity and confidentiality labels reported to them by other agents in the network to collectively compute *global integrity and confidentiality labels*, denoted i_o and c_o respectively, for objects o whose keys fall within that range. Similarly, score-managers with a common key-range collectively compute *global trust values*, denoted t_a, for agents a whose keys fall within that range. Specifically, i_o, c_o, and t_a are defined by

$$i_o := \text{median}\,\{i_{a_{kh}}(o) \mid key_o \in kr(a_{kh})\} \quad (1)$$

$$c_o := \text{median}\,\{c_{a_{kh}}(o) \mid key_o \in kr(a_{kh})\} \quad (2)$$

$$t_a := \text{median}\,\{t_{a_{sm}}(o) \mid key_a \in kr(a_{sm})\} \quad (3)$$

Thus, the global labels for any object o and the global trust value for any agent a can be computed by any agent in the network by contacting all key-holders a_{kh} for object o, or all score-managers a_{sm} for agent a.

3.2. Network Design

A Penny ring is like a Chord ring, with Penny's identifier ranges being equal to Chord's key-ranges. However, a Penny agent's key-range strictly subsumes its identifier range, and agent key-ranges are not unique. Key-ranges are assigned in a Penny ring so that for every agent a, there are between k and $2k$ agents in the ring whose key-ranges are equal to $kr(a)$ (unless the entire network includes less than k total agents), where k is a fixed constant defined at network initialization. Bounding the number of potential key-holders from below by k limits the influence of malicious agents, because it ensures that a single key-holder determines at most $1/k$th of the votes in Equations 1 and 2 that determine an object's global labels, and a single score-manager determines at most $1/k$th of the votes in Equation 3 that determine an agent's global trust value. Bounding the number of potential key-holders from above by $2k$ ensures that lookup will not become too costly, and it bounds the storage overhead for finger tables.

This $O(k)$ redundancy of score-managers is similar to the EigenTrust algorithm [7], but unlike EigenTrust, Penny ensures that all key-holders and score-managers for a given key-range are always located in adjacent positions on the Chord ring. This is achieved by ensuring that each agent's key-range always includes its identifier range. An agent can therefore contact all score-managers for a particular agent a, or all key-holders for a particular object o, using $O(\log N + k)$ messages. The first $O(\log N)$ messages propagate the message using the Chord algorithm [9] to the agent whose identifier range includes key_a or key_o. This agent then forwards the message directly to the other $O(k)$ agents whose key-ranges also include key_a or key_o. Penny therefore reduces the overhead of all network operations that involve contacting key-holders and score-keepers by a factor of k over the EigenTrust algorithm. When k is a large constant, such as $k = 16$, this can mean a significant reduction in network traffic.

To maintain the invariant that the number of score-managers for each key-range lies between k and $2k$, a Penny network must occasionally split or merge key-ranges as agents join and leave the network. If a join operation causes the number of score-managers in a range to rise above $2k$, the Penny protocol splits that key-range into two smaller key-ranges. Dually, if a leave operation causes the number of score-managers for a range to descend below k, Penny reassigns those agents (and some agents in an adjacent key-range) a larger key-range.

Unlike Penny, a Chord network requires each key-holder to maintain a list of the agents who own the key-holder's daughter objects. These lists are reported to any agent who requests the object, divulging the identities of all agents who own a particular object. To address this privacy vulnerability, Penny conceals information associating agents with the objects they own by splitting that information amongst key-holders and score-managers. A malicious key-holder and a malicious score-manager must therefore collaborate to learn that a particular server owns a particular object. Opportunities for such collaboration are limited because key-holders and score-managers cannot choose their key-ranges. It is therefore unlikely that a malicious collective will occupy both a key-range that includes a particular victim object's key and a key-range that includes a particular victim agent's key.

4. The Penny Algorithm

The Penny algorithm describes the key assignment process, agent joins and departures, requesting objects, offering new or downloaded objects with confidentiality and integrity labels, and updating confidentiality and integrity labels as data is exchanged.

4.1. Message Routing

As in Chord, each agent a in a Penny ring maintains a finger table that is used to route messages efficiently. For each $i \in 0 \ldots m-1$, agent a's finger table includes the agent whose identifier range includes $(id_a + 2^i) \bmod 2^m$ (where 2^m is the size of the agent identifier space). In addition, agent a's finger table also includes an entry for each agent whose key-range coincides with a's key-range. We refer to a set of agents with equal key-ranges as a *neighborhood*. The size of each finger table is therefore $O(m+k)$, where k is a constant dictating the number of redundant key-holders assigned to each key.

The protocols for joining and leaving a Penny network are similar to those for joining and leaving a Chord network [9], but Penny's addition of non-unique key-ranges requires special consideration. When an agent a_{new} joins a Penny ring, it is by default assigned a key-range identical to its successor's. Its successor informs all agents in its neighborhood that they should update their finger tables to include a_{new}. However, if this would result in a neighborhood of size greater than $2k$, a split occurs. The first k agents and the last $k+1$ agents in the neighborhood each become their own neighborhoods. The key-ranges of the new neighborhoods are the unions of the identifier ranges of the agents within each.

Figure 1 illustrates a join operation with a split. Identifiers are labeled next to each agent outside the ring, and agent key-ranges are labeled inside the ring. In this example, $k = 2$, so when the agent with identifier 61 joins, key-range $[15, 63]$ has more than $2k$ agents and must be split.

Figure 2 shows an example of the propagation of a Penny message through the resulting ring. Agent 0 wishes to send a message to all agents whose key-range includes identifier 28. First, the message is propagated along the ring according to the Chord algorithm to the agent whose identifier range includes 28 (agent 23). This involves first sending the message to the agent whose identifier range includes $0+2^4 = 16$, and next to the agent whose identifier range includes $16 + 2^3 = 24$ (agent 23). Once the message reaches an agent whose key-range includes 28, that agent forwards the message directly to all other agents in its neighborhood. These are all agents in the ring whose key-ranges include 28.

When an agent a_{old} leaves a Penny ring, it informs its predecessor a_{pred} and the other agents in a_{old}'s neighborhood. If a_{pred} is in a different (adjacent) neighborhood, a_{pred} must inform the other agents in that neighborhood that the neighborhood's key-range has grown to include identifiers up to and including $id_{succ} - 1$ (where a_{succ} is a_{old}'s successor). Likewise, agents in a_{old}'s neighborhood must shrink their key-ranges so that they begin with id_{succ}.

If the departure of a_{old} causes a_{old}'s neighborhood to have fewer than k members, two adjacent neighborhoods must be merged. Let H_{old} and H_{pred} be a_{old}'s and a_{pred}'s neighborhoods, respectively. If $|H_{old}| < k$, then the agent in H_{old} whose predecessor is in H_{pred} sends a merge request to its predecessor. That merge request is then forwarded to all agents in H_{pred}. If $|H_{pred}| \leq k+1$ then both neighborhoods merge to form a single neighborhood. Otherwise, the rightmost $\frac{1}{2}(|H_{pred}| - |H_{old}|)$ agents of neighborhood H_{pred} join neighborhood H_{old}. The key-ranges of the new neighborhoods are the unions of the identifier ranges of the agents in the new neighborhoods.

Figure 3 illustrates an agent leave operation that requires a key-range merge. Here, the departure of agent 15 from the ring leaves fewer than $k = 2$ agents in its neighborhood. Agent 16 therefore merges with its predecessor neighborhood; agents in both neighborhoods extend their key-ranges to include the identifier ranges of all agents in the new neighborhood.

Whenever an agent's key-range shrinks due to any of the above operations, it must transfer any state associated with keys not in its new range to other key-holders. Similarly, whenever its key-range grows, it receives state associated with new keys from the agents who previously occupied that range. On average, $k/2$ agents must join or leave a neighborhood before that neighborhood will need to be split or merged. Thus, by initializing k to a large constant, the frequency of these state transfer operations can be reduced.

4.2. Agent Local State

In addition to routing messages, each agent in a Penny network plays three different roles: It acts as a server when sharing objects, it acts as a score-manager for agents whose keys fall within its key-range, and it acts as a key-holder for objects whose keys fall within its key-range. For each of these roles, the agent maintains some internal state:

- To act as a server, an agent a maintains a list of the identifiers id_o, and local integrity and confidentiality labels $i_a(o)$ and $c_a(o)$ of each object o that it owns. The agent also chooses a public key, private key pair (K_a, k_a).

- To act as a score-manager, agent a maintains a list of daughter agents a_d that satisfy $key_{a_d} \in kr(a)$. These are the agents for whom agent a is a score-manager. For each daughter agent a_d, agent a maintains a vector of local trust values $t_{a'}(a_d)$ reported by the agents a' that have interacted with agent a_d.

- To act as a key-holder, agent a maintains a list of the identifiers of daughter objects o that satisfy $key_o \in kr(a)$. These are the objects for which agent a is a key-holder. For each such object o, agent a maintains

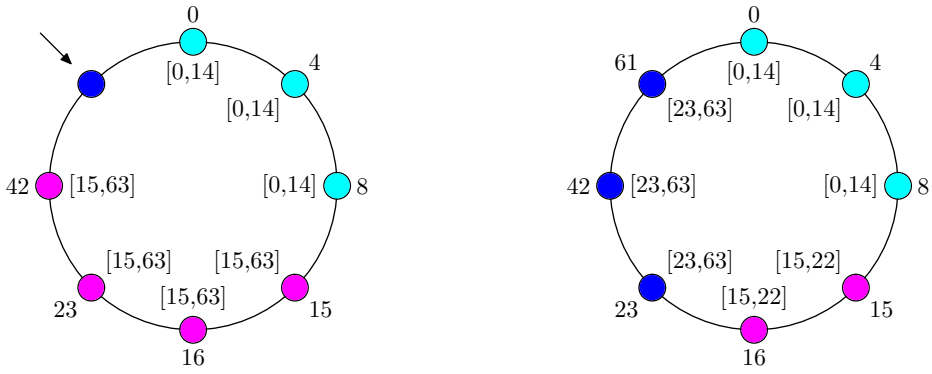

Figure 1. Agent join

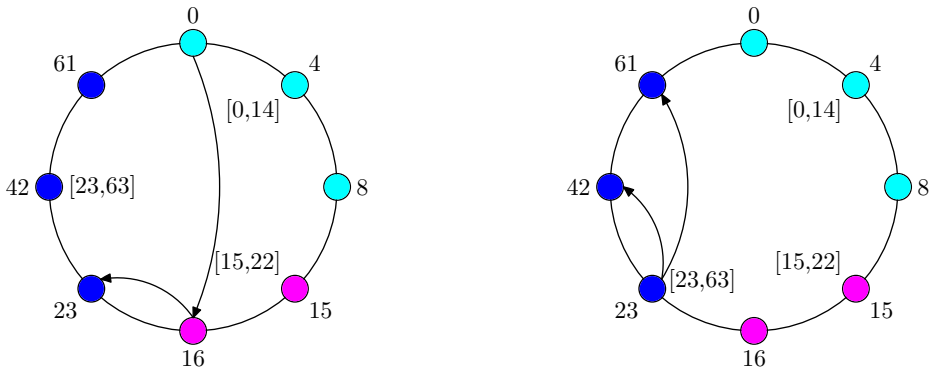

Figure 2. Message propagation

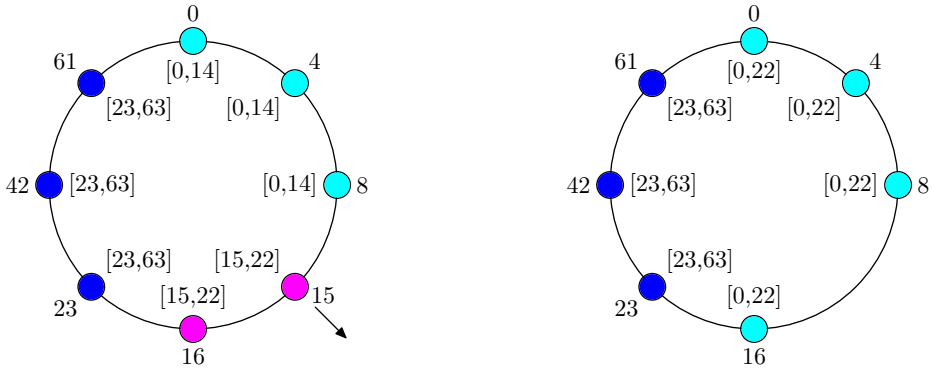

Figure 3. Agent leave

a vector of integrity labels $i_{a'}(o)$ and confidentiality labels $c_{a'}(o)$ reported by the agents a' that have reported feedback about object o. Agent a also maintains a list of the keys key_{svr} and public keys K_{svr} of agents that serve object o. Key-holders do not learn the actual identifiers of agents a' or a_{svr}, only their keys.

4.3. Publishing and Downloading Objects

Once a Penny network has been initialized, agents interact according to the protocol detailed below. The protocol diagrams that follow use solid arrows to denote messages that are sent directly from agent to agent without using the P2P overlay, and dashed arrows for messages that use the P2P overlay to find the message target based on its ring identifier. Dashed arrows therefore actually entail sending $O(\log N+k)$ total messages, where the first $O(\log N)$ messages are sent in series along the ring and the last $O(k)$ messages are sent in parallel to the other agents in the receiving agent's neighborhood (see §4.1 for an example). Arrows with double-heads could be sent anonymously—e.g., via an anonymizing tunnel [22, 23]. Notation K_a denotes agent a's public key, and $\langle \ldots \rangle_K$ denotes a message encrypted with key K. Notation $\Delta t_{svr}(a)$ denotes feedback provided by agent a_{svr} after a particular (possibly anonymous) transaction with agent a. Such feedback is a boolean value indicating whether the transaction was acceptable.

When an agent a_{svr} wishes to share an object o, it must first *publish* that object according to the protocol depicted in Fig. 4. Agent a_{svr} first obtains (possibly anonymously) the public keys of all key-holders a_{kh} for object o. (Recall that score-managers for object o are defined as those agents whose key-ranges include key_o.) Agent a_{svr} next encrypts the object identifier, local integrity and confidentiality labels, and its own public key, with each of the key-holders' public keys. It asks one of its own score-managers a_{sm} to forward the encrypted messages to the key-holders a_{kh}. (Recall that score-managers for agent a_{svr} are defined as those agents whose key-ranges include key_{svr}.) Agent a_{sm} conceals agent a_{svr}'s identity by sending only its key to the key-holder rather than its identifier, along with global trust value t_{svr} (computed via Equation 3).

Each key-holder then updates its local integrity and confidentiality labels for o according to the formulas

$$i_{kh}(o) := \frac{\sum_{a \in A}(t_a \cdot i_a(o))}{\sum_{a \in A} t_a} \tag{4}$$

$$c_{kh}(o) := \frac{\sum_{a \in A}(t_a \cdot c_a(o))}{\sum_{a \in A} t_a} \tag{5}$$

where A is the set of agents (including a_{svr}) from whom key-holder a_{kh} has received a local integrity and confidentiality labels for object o. Note that key-holders will not generally know the identifiers of agents in set A; they will only know their keys, which is enough information to obtain trust value t_a from each agent's score-managers.

To request an object (Fig. 5), requester a_{req} first sends the requested object's identifier to all key-holders a_{kh} for the object. Each key-holder responds with the object's integrity label, the object's confidentiality label, and a list of the keys and public keys of the servers who offer the object. Agent a_{req} can then obtain the object from any server a_{svr} by sending a request to all score-managers for agent a_{svr}. In the request message, the identifier for the requested object is encrypted with the server's public key to avoid disclosing it to the agent score-managers. The score-managers forward the request to the server. The server can then anonymously send the data directly to the requester. Once the transfer is complete, the requester may report feedback on the transaction to agent a_{svr}'s score-managers. The requester can also report feedback on the integrity and confidentiality of the data it received by following the publish protocol (Fig. 4). (An obvious simplification of the publish protocol can also be used to report feedback without publishing the object.)

Score-managers maintain a vector of local trust values $t_a(a_d)$ for each of their daughter agents a_d. They use this vector in accordance with the EigenTrust algorithm [7] to compute a global trust value t_{a_d} for each daughter agent.

5. Discussion

Penny inhibits the spread of low-integrity data (e.g., malware) by maintaining a centralized integrity label for each object shared over the network. Agents wishing to avoid such data can therefore consult each object's integrity label before downloading it. Thus, the problem of restraining the spread of malware over a Penny network reduces to the problem of efficiently maintaining and reporting accurate integrity labels.

Label retrieval is efficient in Penny, requiring approximately the same number of messages as object lookup in a Chord network. An agent can retrieve any object's integrity label by sending a single request message, which gets forwarded at most $O(\log N+k)$ times throughout the network. The request solicits $O(k)$ response messages, which are aggregated to compute the centralized integrity label (see Equation 1).

An object's centralized integrity label is determined by the votes of other agents in the network (see Equation 4). Votes are weighted by the reputation of each voter so that the votes of agents who are widely regarded as trustworthy are more influential than the votes of those who are not. This makes it difficult for a malicious agent to attach a high-integrity label to low-integrity data. In order for such an attack to succeed, malicious agents must collectively have such good reputations that they outweigh the votes of all

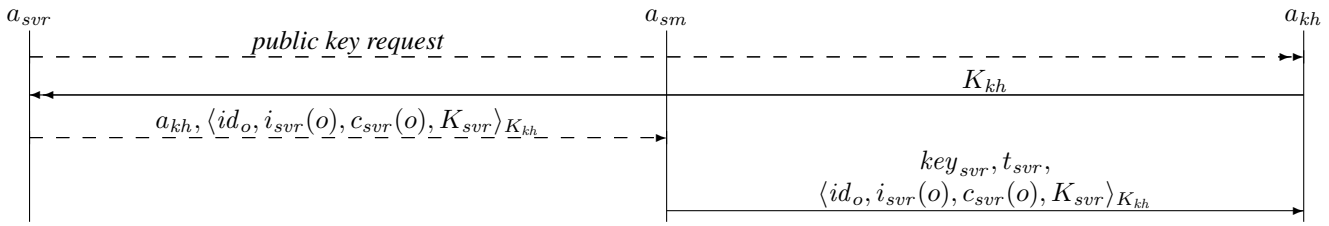

Figure 4. Publish protocol.

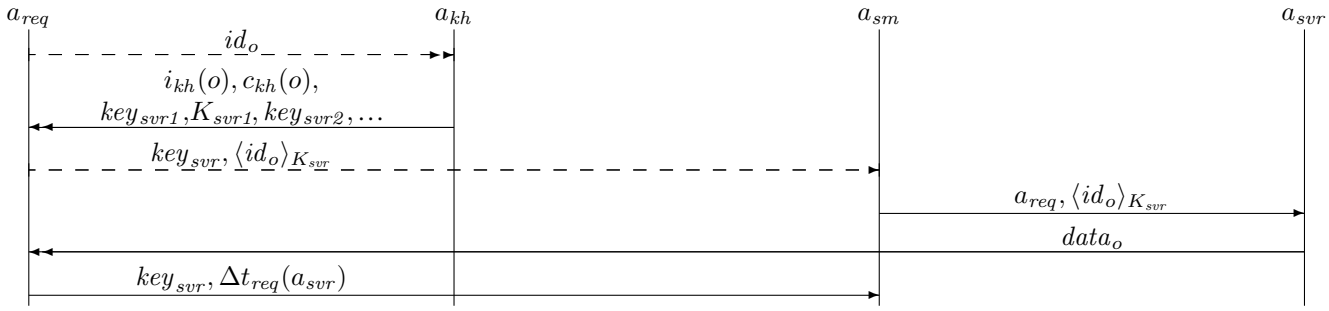

Figure 5. Request protocol.

other voters. Penny uses the EigenTrust algorithm [7] to track agent reputations and to prevent malicious agents from accruing good reputations.

Penny employs both secure hashing and replication to protect against malicious key-holders and score-managers who might falsify an object's centralized integrity labels or an agent's centralized trust value. Use of a secure hash function for identifier assignment ensures that agents cannot dictate the set of objects and agents for which they serve as key-holders and score-managers. By ensuring that there exist at least k key-holders and score-managers for every key-range, Penny prevents any one agent from subverting the reputation of any object or agent. At least $k/2$ agents in a neighborhood must be malicious in order to subvert a reputation.

Malicious peers cannot elevate their own reputations by switching IP addresses or creating false network accounts because, as in EigenTrust, all agent and object reputations start at zero in Penny. A peer or object acquires a positive reputation only by participating in positive transactions with other peers. Peers with established reputations then report positive feedback for those transactions, elevating the new peer's reputation.[1] Changing IP addresses or creating a new account therefore never results in an increase to the peer's reputation.

The protocol described in §4.3 also enforces a notion of object ownership privacy by separating information linking a server to the objects it owns. Specifically, key-holders for an object learn only the keys of servers who own the object but not the identifiers themselves, whereas score-managers learn the identities of the servers but not the identifiers of the objects they own. A malicious score-manager a_{sm} and a malicious key-holder a_{kh} can cooperate to learn that a particular server a_{svr} owns a particular object o, but only if $key_{svr} \in kr(a_{sm})$ and $key_o \in kr(a_{kh})$ both hold, which is unlikely when the size of a malicious collective is not significant relative to the size of the network.

Key-holders and score-managers can, of course, learn ownership information through guessing attacks, but this is prohibitively expensive when the space of object and agent identifiers is large. For example, a malicious agent a_m can discover if a particular object o is served by any agent for which a_m serves as score-manager by requesting id_o and comparing the key-holders' responses against its list of daughter agents. However, a_m cannot easily produce a list of all objects served by any of its daughter agents because to do so it would have to search the entire space of object identifiers. Likewise, a_m can discover if a particular server a_{svr} owns any object for which a_m serves as key-holder. To do so, a_m computes key_{svr} and searches for that key in its list of keys of servers that own a_m's daughter objects. However, a_m cannot easily produce a list of all servers that own any given object because it would have to search the entire space of server identifiers.

In addition to integrity labels, Penny also maintains centralized confidentiality labels for objects. Agents can use these labels as a basis for selectively serving data to other

[1] This obviously means that if all peers in the network have reputation zero, no reputations can be elevated. To prevent this, a root set of *network founders* start with reputation 1 at network initialization.

peers—possibly based on the requester's trust level or other credentials. However, to enforce stronger confidentiality policies, such as mandatory access control policies that prohibit low-trust agents from obtaining high-confidentiality data, agents need a means to detect confidentiality violations and report them to the trust management system. This is necessary to force a misbehaving agent's trust value to decrease, preventing future violations. Detecting confidentiality violations is a challenging research problem, and is discussed in §6 as part of future work.

6. Conclusion and Future Work

Penny combines reputation-based trust management (based on EigenTrust [7]), distributed hash tables (based on Chord [9]), and anonymizing tunnels (based on Tarzan [22, 23]) to support secure integrity and confidentiality labeling of shared data as well as ownership privacy for object servers. The protocol enforces these security guarantees in an efficient manner that avoids broadcast messages or large-scale polling of peers. Object lookup can be performed with $O(\log N + k)$ messages, where N is the number of agents in the network and k is a constant that controls how much replication is used to protect against malicious agents.

We are in the process of implementing Penny client software in Java. This ongoing research is aimed at evaluating Penny in a practical setting to verify that the network structure is (i) efficient enough to handle realistic P2P network traffic, and (ii) robust enough to prevent malicious collectives from subverting integrity labels, confidentiality labels, and trust values.

Future research should also consider how to enforce various information flow and access control policies based on Penny's integrity and confidentiality labeling system. For example, Penny's publish and request protocols might be augmented with security checks that block the dissemination of objects whose integrity labels lie below a certain threshold. This would have the effect of censoring known malware from the network.

One might also enforce a corresponding confidentiality policy that prohibits low-trust agents from obtaining high-confidentiality data, but this is a more difficult research challenge. In order to prevent future confidentiality violations, the trust management system must be informed of past confidentiality violations. It is unclear how to ensure that past confidentiality violations get reported, since typically the only witnesses of these violations are the malicious agents involved in leaking the data. Enforcing strong confidentiality policies in P2P networks therefore remains an interesting open problem.

Our analysis did not consider attacks upon the P2P network overlay itself, such as denial of service, message misrouting, message tampering, or traffic pattern analysis. Using trust values to change the routing structure (so as to avoid routing messages through malicious agents) is an interesting and active area of research that might address these vulnerabilities. In addition, our network remains vulnerable to certain Sybil attacks [24], wherein an individual malicious agent masquerades as many different agents with different identifiers in an effort to control a large percentage of the identifier space, or to cast multiple votes. Future work should investigate augmenting the EigenTrust algorithm with IP address clustering and other techniques aimed at identifying malicious collectives that are actually comprised of many pseudonyms of a single peer.

Finally, Penny is one contribution to the larger research question of how to combine anonymity with reputation-based trust management. Anonymity and reputation-based trust are often at odds because it is difficult to divulge an agent's reputation without also divulging its identity. Penny illustrates one way to communicate and evaluate the reputation of an agent without fully disclosing its identity; however, divulging the agent's reputation might in some cases reduce the anonymity of a transaction (e.g., in cases where the attacker is powerful enough to search the space of all agent reputations in the network for a match). Development of more general-purpose algorithms for evaluating the trustworthiness of a message-sender without knowing its identity would allow P2P networks to safely enforce stronger privacy policies in decentralized settings.

References

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

[2] Gnutella. http://www.gnutella.com

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

[4] Limewire. http://www.limewire.com

[5] Shin, S., Jung, J., Balakrishnan, H.: Malware prevalence in the KaZaA file-sharing network. In: Proc. 6th ACM SIGCOMM Internet Measurement Conf. (IMC'06), Rio de Janeiro, Brazil (October 2006) 333–338

[6] Kalafut, A., Acharya, A., Gupta, M.: A study of malware in peer-to-peer networks. In: Proc. 6th ACM SIGCOMM Internet Measurement Conf. (IMC'06), Rio de Janeiro, Brazil (October 2006) 327–332

[7] Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The EigenTrust algorithm for reputation management in P2P networks. In: Proc. 12th Int. World Wide Web Conf. (WWW'03), Budapest, Hungary (May 2003) 640–651

[8] Damiani, E., di Vimercati, S.D.C., Paraboschi, S., Samarati, P., Violante, F.: A reputation-based approach for choosing reliable resources in peer-to-peer networks. In: Proc. 9th ACM Conf. on Comp. and Comm. Sec. (CCS'02), Washington, DC (November 2002) 207–216

[9] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proc. ACM Conf. on Applications, Technologies, Architectures, and Protocols for Comp. Comm. (SIGCOMM'01), San Diego, California (August 2001) 149–160

[10] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable, content-addressable network. In: Proc. ACM Conf. on Applications, Technologies, Architectures, and Protocols for Comp. Comm. (SIGCOMM'01), San Diego, California (August 2001) 161–172

[11] Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In: Proc. IFIP/ACM Int. Conf. on Distributed Sys. Platforms (Middleware '01), Heidelberg, Germany (November 2001) 329–350

[12] Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.D.: Tapestry: A resilient global-scale overlay for service deployment. IEEE J. on Selected Areas in Comm. (JSAC'04) **22**(1) (January 2004) 41–53

[13] Aberer, K., Despotovic, Z.: Managing trust in a peer-2-peer information system. In: Proc. 10th ACM Int. Conf. on Information and Knowledge Management (CIKM'01), New York (November 2001) 310–317

[14] Zacharia, G., Maes, P.: Trust management through reputation mechanisms. Applied Artificial Intelligence **14**(9) (October 2000) 881–907

[15] Xiong, L., Liu, L.: PeerTrust: Supporting reputation-based trust in peer-to-peer communities. IEEE Trans. on Knowledge and Data Engineering (TKDE'04) **16**(7) (July 2004) 843–857

[16] Lee, S., Sherwood, R., Bhattacharjee, B.: Cooperative peer groups in NICE. In: Proc. 22nd Annual Joint Conf. of the IEEE Comp. and Comm. Soc. (INFOCOM'03), San Francisco, California (April 2003) 1272–1282

[17] Gupta, M., Judge, P., Ammar, M.H.: A reputation system for peer-to-peer networks. In: Proc. 13th ACM Int. Workshop on Network and Op. Sys. Support for Digital Audio and Video (NOSSDAV'03), Monterey, California (June 2003) 144–152

[18] Blaze, M., Feigenbaum, J., Strauss, M.: Compliance checking in the PolicyMaker trust management system. In: Proc. 2nd Int. Financial Cryptography Conf. (FC'98), Anguilla, British West Indies (February 1998) 254–274

[19] Marsh, S.: Formalising Trust as a Computational Concept. PhD thesis, University of Stirling, Scotland, UK (April 1994)

[20] Sabater, J., Sierra, C.: Reputation and social network analysis in multi-agent systems. In: Proc. 1st ACM Int. Joint Conf. on Autonomous Agents and Multiagent Sys. (AAMAS'02), Bologna, Italy (July 2002) 475–482

[21] Pujol, J.M., Sangüesa, R., Delgado, J.: Extracting reputation in multi-agent systems by means of social network topology. In: Proc. 1st ACM Int. Joint Conf. on Autonomous Agents and Multiagent Sys. (AAMAS'02), Bologna, Italy (July 2002) 467–474

[22] Freedman, M.J., Sit, E., Cates, J., Morris, R.: Introducing Tarzan, a peer-to-peer anonymizing network layer. In: Proc. 1st Int. Conf. on Peer-to-peer Sys. (IPTPS'02), Cambridge, Massachusetts (March 2002) 121–129

[23] Freedman, M.J., Morris, R.: Tarzan: A peer-to-peer anonymizing network layer. In: Proc. 9th ACM Conf. on Comp. and Comm. Sec. (CCS'02), Washington, DC (November 2002) 193–206

[24] Douceur, J.R.: The Sybil attack. In: Proc. 1st Int. Workshop on Peer-to-peer Sys. (IPTPS'02), Cambridge, MA (March 2002) 251–260