Penny: Secure, Decentralized Data Management

Safwan Mahmud Khan and Kevin W. Hamlen
(Corresponding author: Safwan Mahmud Khan)

Computer Science Department, The University of Texas at Dallas
800 W. Campbell Rd., Richardson, Texas 75080
(Email: {safwan,hamlen}@utdallas.edu)
(Received Nov. 20, 2012; revised and accepted May 4, 2013)

Abstract

Structured, peer-to-peer (P2P) networks scalably support efficient data-retrieval operations over large data sets without resorting to centralized servers that may be vulnerable to attack or failure. However, data confidentiality policies, such as data ownership privacy, are difficult to enforce reliably in these networks. This paper presents Penny, a structured P2P network that efficiently supports integrity and confidentiality labeling of data, and enforces a notion of ownership privacy that permits peers to publish data without revealing their ownership of the data. A decentralized reputation management system allows the network to respond and adapt to malicious peers. These features are applied to securely manage traditional files and large RDF data sets stored as N-triples in a P2P environment. Simulations demonstrate that Penny can efficiently handle realistic P2P network traffic and robust enough to prevent malicious collectives from subverting security labels.

Keywords: Peer-to-peer security, Reputation management, Trust management, Distributed computing, RDF data

1 Introduction

Peer-to-peer (P2P) networking is a distributed, load-balancing computing paradigm designed to scalably share work loads between peers. Unlike traditional client-server models, each peer in a P2P network is an equally privileged, equipotent participant in the distributed computation or service. This has the advantage of avoiding centralized points of failure that, when successfully attacked, suffice to dismantle the entire network. P2P was first popularized as a vehicle for music-sharing [28], but has since expanded to general-purpose file- and data-sharing applications (e.g., [4, 14, 21]) and is increasingly important as a basis for fault-tolerant cloud computing [26]. Since its inception, it has been tremendously popular and ubiquitous because of its collective computation power, natural load-balancing, and low-cost deployability. For example, it has been estimated that BitTorrent traffic accounts for roughly 27–55% of all Internet traffic (depending on geographical location) as of February 2009 [35].

However, while P2P networks have proven successful for maintaining high data availability under adversarial or noisy conditions, enforcing strong data integrity and confidentiality under these conditions remains a difficult challenge. Integrity enforcement is challenging because P2P networks lack a centralized authority who can identify and evict malicious nodes from the network. Malicious nodes are therefore free to propagate malicious code or untrustworthy data by misrepresenting it as a high-integrity resource available for download [39]. Approximately 18.5% of all BitTorrent downloads contain malware [3] as a result. Confidentiality enforcement is impeded by the explicit divulgence of sender peer positions during overlay communications while the requesting peer remains anonymous [6, 9]. This allows malicious peers to anonymously identify and target purveyors of security-relevant resources.

To address these deficiencies, we have designed, implemented, and tested Penny, a P2P networking protocol that extends Chord [36] with secure integrity- and confidentiality-labeling of shared data. Penny uses a distributed reputation management system based on EigenTrust [20] to securely manage data labels without the introduction of a central authority. The data labels empower requester peers to avoid downloads of low-integrity data, and allow sender peers to deny low-privilege peers access to high-confidentiality data. In addition, sender peers may publish and serve their data anonymously, frustrating attacks that seek to single out and target owners of security-relevant data.

We have applied Penny to construct a secure, fully decentralized, data management system for traditional data files as well as Resource Description Framework (RDF) data. RDF is a popular web data format that is of particular importance to Semantic Web technologies. Stores of RDF data can be extremely large and security-sensitive, resulting in an increasing demand for secure distributed computing paradigms that can manage them efficiently (cf., [8, 22]). Managing RDF data in a P2P network has the advantage of facilitating dynamic, low-cost growth of the network to accommodate expansion of the data set without sacrificing the security guarantees traditionally associated with more centralized networks, such as clouds. Though there has been much research within the semantic

web community on the security and privacy of RDF data, few works consider a fully decentralized approach like the one considered here.

Our prior work proposed Penny's overlay and resource-sharing protocol as a preliminary study without any implementation or experiments [37]. We here extend that theoretical work with improvements to the architectural design, new formulas for computing data integrity and confidentiality labels, empirically determined optimal neighborhood sizes, new publish and request protocols adapted for RDF data, and other new empirically tested algorithms necessary for the system. We also describe a full implementation of the Penny client, supporting traditional files and RDF datasets, with experimental results and analysis.

Related works are first summarized in §2. Penny's architectural design, including publish and download protocols, is then presented in §3. We describe our simulation methodology, results, and analysis in §4. Security properties enforced by the design are discussed in §5. Finally, we conclude with suggestions for future work in §6.

2 Related Work

P2PRep [10] is one of the first works to implement secure reputation management in a real-world P2P network [14]. Resource-requesting peers in the network assess the reliability of perspective providers before initiating downloads by polling large numbers 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 depend on which peers 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.

XRep [11] enhances P2PRep by combining reputations of providers and resources, offering more informative polling and overcoming the limitations of strictly provider-based solutions. However, it still suffers the disadvantages of P2PRep above. ServiceTrust [19] is a service-oriented paradigm that computes trust globally, but at the cost of centralizing the system, inviting centralized points of failure. One solution is to compute trust globally using decentralized gossip-based algorithms [2]. However, this does not recursively apply the trust system to the gossip itself, allowing malicious agents to potentially gain undue influence by reporting high trust for malicious allies.

In contrast to these unstructured approaches, Penny is implemented atop a structured P2P protocol—Chord [36]. Chord solves the fundamental problem of efficiently locating peers with particular data objects by assigning a unique identifier to each peer, and arranging them 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 peers in the network. In Chord, 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 [32], Pastry [33], Tapestry [42], and MAAN [7]. These systems offer distributed, scalable, and efficient search, but they do not include data security or privacy enforcement mechanisms. Penny extends Chord by providing a framework for maintaining centralized security labels for data shared over a Chord network.

In trust management systems, peers and occasionally objects in the system are labeled with trust values based on past peer interactions. These past experiences are consulted to predict future malicious behavior and incentivize good behavior. 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. Examples include EigenTrust [20], DMRep [1], P2Prep [10], XRep [11], Sporas and Histos [41], PeerTrust [40], NICE [23], and DCRC/CORC [17]. In contrast, *policy-based* trust management systems, such as PolicyMaker [5], derive peer trust based on supplied credentials. Finally, trust management systems based on *social networks* determine trust by analyzing a complex social network. Examples include Marsh [27], Regret [34], and NodeRanking [31].

Penny integrates a reputation-based trust management system based on secure EigenTrust [20]. Each peer is assigned a global trust value based on the peer's history of downloads. Global trust values are computed in a distributed manner with minimal load, resulting in assured convergence for all trust queries without centralization.

Resource Description Framework (RDF) is a metadata model for web data exchange. It is widely used for semantic web knowledge due to its expressive power, semantic interoperability, and reusability. We show that Penny is well-suited not only for traditional P2P file/object lookups/downloads, but also for deploying and querying RDF datasets. Two categories of prior work have investigated effective RDF data management in P2P environments. One considers the problem of distributing and retrieving RDF data efficiently [8, 29, 30], while the other proposes algorithms for efficient query processing (but not storage) [24, 25, 38]. Neither body of work considers peer trust or data security issues to our knowledge.

Penny's RDF representation scheme is most closely related to that of RDFPeers [8]. RDFPeers stores each

RDF triple by hashing it three times (once for subject, predicate, and object, respectively), resulting in three replicas. Penny adopts a similar strategy, but for more efficient query processing it indexes each query by hashing only one of the three parts, retrieves a superset of the desired triples, and locally filters the results.

To ensure confidentiality, Penny peers must send some messages anonymously. This is accomplished via anonymizing tunnels [12, 13, 43], which permit peers to route their messages through tunnels of randomly chosen peers. Multilayer encryption and randomly generated cover traffic prevent any peer in the tunnel from learning whether its successor is the message originator or just another hop in the tunnel. The tunnels are bidirectional, allowing recipients 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 [12, 13].

3 Penny Network

Penny implements a standard Chord ring, but with an extended form of reputation tracking: For each peer and data object, Penny allots k *score-manager* peers and k *key-holder* peers (respectively) to compute and track the peer or object's trust label(s). Parameter k is fixed at network start and controls the degree of replication; greater k means greater security, since attackers must compromise more peers to successfully corrupt data. Penny strategically positions responsibility-sharing score-managers and key-managers at adjacent ring positions, forming a *neighborhood*. This greatly improves lookup efficiency over standard Chord, since only one overlay message (instead of k) suffices to contact all k replicas. The result is high replication (and therefore high security) with low overhead.

To protect data ownership privacy, data lookups in Penny employ a cryptographically protected extra level of indirection. Data-serving peers first encrypt their requests with the public key of the data item's key-holder, and then ask an arbitrary score-manger to forward the server's key (not its real identifier) and encrypted information to the key-holder. As a result, the key-holder does not know who the real owner of the data item is, and so when someone later requests that data item, the key-holder forwards the request back through the score-manger(s). Meanwhile, the score-mangers do not know which data items are owned by which peers, and thus learn no peer-object associations as they forward the requests. As a result, the ownership information is concealed from all other parties.

We explain this protocol in detail below, beginning with foundational definitions in §3.1 and proceeding to architectural details in §3.2.

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 identifiers 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 $(id_{pred(a)}, id_a]$ 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 obtained by applying a one-way, deterministic hash function to its 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.

Key-holders and score-managers: Each agent a_1 is assigned a (not necessarily unique) *key-range*, denoted $kr(a_1)$. Agent a_1 is charged with tracking the global integrity and confidentiality labels (discussed later) assigned to all objects o that satisfy $key_o \in kr(a_1)$. In addition, agent a_1 tracks the global trust values (discussed later) 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.

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. Similarly, there are local integrity and confidentiality labels for agents with whom other agents had transactions. 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 or object 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. Likewise, each ordered pair of agent and object (a, o) has a *local trust value* denoted $t_a(o)$ that measures the degree to which agent a trusts object o. Like confidentiality and integrity labels, trust values range from 0 to 1 inclusive (cf., [20]). Local integrity and confidentiality labels are computed and assigned based on local trust values.

Global labels and trust values: Each object o in the system is associated with global integrity and confidentiality labels, denoted i_o and c_o, respectively, and measured by global trust values T_o. Likewise, each agent a is associated with global integrity and confidentiality labels, denoted i_a and c_a, respectively, and measured by global trust values T_a. Key-holders with a common key-range compute T_o and score-managers with a common key-range calculate T_a using Secure EigenTrust [20]. Thus, the global labels and global trust values for any object o and for any agent a can be acquired by any agent in the network by contacting all key-holders a_{kh} for object o, and all score-managers a_{sm} for agent a.

3.2 Network Architecture

3.2.1 Identifier Space and Neighborhood

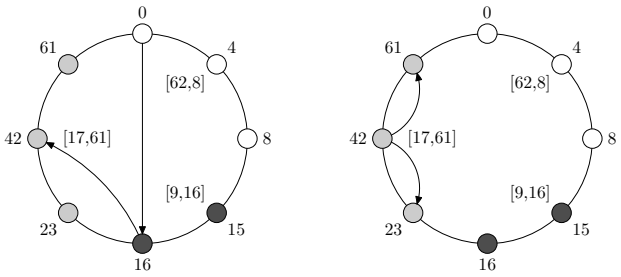
A Penny ring is like a Chord ring, with Penny's identifier ranges being equal to Chord's key-ranges. However, a Penny agent's key-range strictly subsumes its identifier range, and agent key-ranges are not unique. Key-ranges are assigned in a Penny ring so that for every agent a, there are between $\min(k, n)$ and c agents in the ring whose key-ranges are equal to $kr(a)$, where n is the total number of agents and c is a fixed bound on neighborhood size. (The choice of c is discussed in §4.1; usually $c = 3k$.) Bounding neighborhood size from below by k limits the influence of malicious agents, because each contributes at most $1/k$ of the responses to a secure query. Bounding it from above by c ensures that lookup is not too costly, and it bounds the storage overhead for finger tables.

3.2.2 Message Routing

An agent can 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 [36] to an agent whose identifier range includes key_a or key_o, who then forwards it directly to the other $O(k)$ agents in its neighborhood. Penny therefore reduces the overhead of all network operations that involve contacting key-holders, score-managers, and RDF data owners by a factor of k over EigenTrust. This permits higher replication rates (e.g., $k = 16$) that are often infeasible with past approaches.

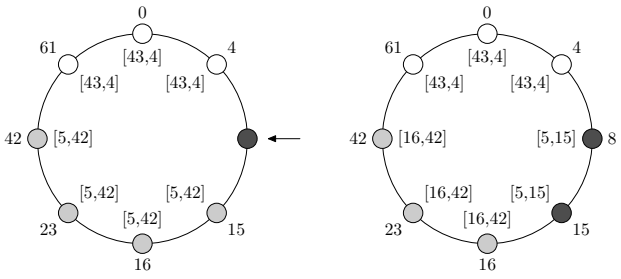
As in Chord, each agent a in a Penny ring maintains a finger table that is used to route messages efficiently. For each $i \in [0, m)$, agent a's finger table includes the agent whose identifier range includes $(id_a + 2^i) \bmod 2^m$ (where 2^m is the size of the identifier space). In addition, agent a's finger table also includes an entry for each agent in its 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.

Figure 1 shows an example of the propagation of a Penny message through the resulting ring. In this example, $m = 6$. Agent 0 wishes to send a message to all agents whose key-range includes identifier 28. First, the message



Step 1: find neighborhood Step 2: broadcast

Figure 1: Penny message propagation



Before After

Figure 2: Agent join operation

is propagated along the ring according to the Chord algorithm to the agent whose identifier range includes 28 (agent 42). This involves first sending the message to the agent whose identifier range includes $0 + 2^4 = 16$ (owner is agent 16), and next to the agent whose identifier range includes $16 + 2^3 = 24$ (owner is agent 42). 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.

3.2.3 Network Dynamics

To maintain the invariant that the number of score-managers for each key-range stays between k and c, a Penny network must occasionally split or merge neighborhoods as agents join and leave the network. If a peer-join causes a neighborhood's population to rise above c, it splits into two smaller neighborhoods. Dually, if peer-leave reduces a neighborhood's population below k, some or all peers from an adjacent neighborhood migrate in.

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 size b that is greater than c, a split occurs. The first $\lfloor b/2 \rfloor$ agents and the last $b - \lfloor b/2 \rfloor$ 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 2 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, $k = 2$, $c = 4$, and $m = 6$, so when the agent with

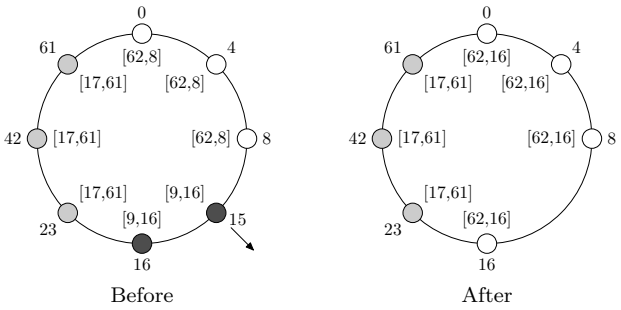


Figure 3: Agent leave operation

identifier 8 joins, key-range $[5, 42]$ has more than c agents and must be split.

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

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} be a_{old}'s neighborhood and H_{pred} be the preceding neighborhood. 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 $(|H_{pred}| - |H_{old}|)/2$ 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 the appropriate key-holders. Similarly, whenever its key-range grows, it receives state associated with new keys from the agents who previously occupied that range. An average net population change of $\frac{1}{2}(c - k)$ agents per neighborhood is required before that neighborhood will need to be split or merged. Thus, by initializing c to be large relative to k, the frequency of these state transfer operations can be reduced.

3.2.4 Agent's Local State

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

- To act as server, it maintains a list of the identifiers id_o of each object o that it owns.
- To act as score-manager, it 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, it also maintains a vector of global trust values T_{a_d} with global integrity and confidentiality labels i_{a_d} and c_{a_d}, respectively.
- To act as key-holder, it maintains a list of daughter objects o_d that satisfy $key_{o_d} \in kr(a)$. These are the objects for which agent a is a key-holder. For each daughter object o_d, it maintains a vector of global trust values T_{o_d} with global integrity and confidentiality labels i_{o_d} and c_{o_d}, respectively.
- For encrypted communication, it chooses a public key, private key pair (K_a, k_a).
- It maintains a list of the keys key_{svr} and public keys K_{svr} of the agents that serve object o. Thus key-holders do not learn the actual identifiers of agents who serve object o, only their keys.
- It maintains local trust values $t_a(a_1)$ and $t_a(o)$ for agents a_1 and objects o with whom it had experience. These local trust values give rise to local integrity and confidentiality labels that agent a associates with a_1 and o.

3.2.5 Publishing and Downloading Protocols for Traditional File Objects

Once a Penny network has been initialized, agents interact according to the protocols 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 involve sending $O(\log N + k)$ total messages. Arrows with double-heads may optionally be sent via anonymizing tunnels for privacy [12, 13, 43]. Notation K_a denotes agent a's public key, and $\langle \ldots \rangle_K$ denotes a message encrypted with key K.

When an agent a_{svr} wishes to share an object o, it must first publish that object according to the protocol depicted in Figure 4. Agent a_{svr} first obtains (possibly anonymously) the public keys of all key-holders a_{kh} for object o. Agent a_{svr} next encrypts the object identifier and its own public key with each of the key-holders' public keys. It asks one of its score-managers a_{sm} to forward the encrypted messages to the key-holders a_{kh}. Agent a_{sm} conceals agent a_{svr}'s identity by sending only its key (which later can be used to get the global trust values and labels from the server's score-manger) to the key-holder rather than its identifier, along with the encrypted message.

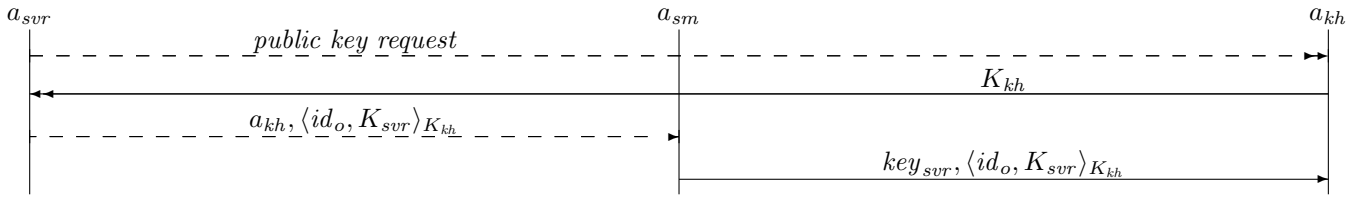


Figure 4: Publish protocol for traditional file objects

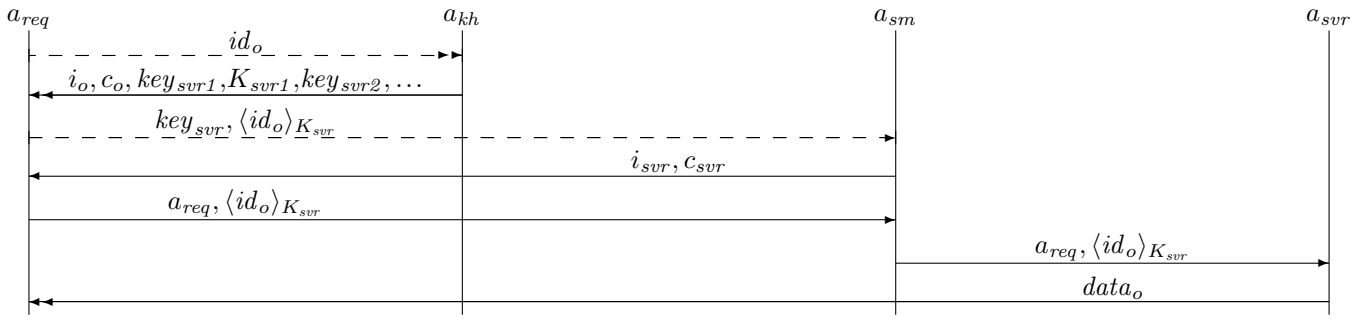


Figure 5: Request protocol for traditional file objects

To request an object (Figure 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 global integrity and confidentiality labels, and a list of the keys and public keys of 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}. Score-mangers reply to a_{req} with the server's global trust labels. Based on a selection procedure (§3.2.8), a_{req} then sends a download request message. In the message, the requested object's identifier is encrypted with the server's public key to avoid disclosing it to the selected server's score-manager. The score-managers forward the request to the server. The server can then anonymously send the data directly to the requester.

3.2.6 Publishing and Downloading Protocols for RDF Datasets

Besides traditional file lookup, Penny also supports RDF dataset queries. RDF triples are stored within file objects, but it would be prohibitively inefficient and insecure to store all triples within a single file owned by a single peer. Triples are therefore distributed over many smaller files distributed across many peers, with a protocol for locating each triple's containing file. Neighborhoods therefore collaborate to manage a subset of triples. Instead of keys for files, we associate triples with identifiers directly, and all neighborhood agents reply with list of servers who own the identified triples. One is chosen from this list using the selection procedure in §3.2.8.

This publishing procedure is detailed in Algorithm 1. To distribute the load of serving particularly popular triples, each agent maintains a usage count for each triple it serves. When this count exceeds an agent-imposed popularity threshold, it defers storage of future instances of that triple component to its successors in the ring. This implements a form of coalesced chaining in the distributed hash table.

RDF queries have syntax `([?]s,[?]p,[?]o)`, where s is a subject, p is a predicate, and o is an object, and where each optional `?` indicates an unknown in the query. For example, query `(s,p,?o)` requests all RDF triples satisfying subject `s` and predicate `p`. For downloading or querying RDF datasets over Penny network, agents implement Algorithm 2.

3.2.7 Reputation-based Trust Management

Penny incorporates a reputation-based trust management system based on EigenTrust [20]. EigenTrust is a secure, distributed trust management system that maintains a globalized trust value for each agent. 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 agent a_i trusts agent a_j, then the left eigenvector v of matrix T measures each agent a's global trust based on how much each agent trusts a, how much each agent trusts the agents who trust a, etc.

If an agent a_i downloads a file or RDF data from an agent a_j, it rates the transaction as positive (rating 1) or negative (rating -1) based on the experience. We may define local trust value $s(a_i, a_j)$ as the sum of these ratings of agent a_j by agent a_i. Then, in order to aggregate the local trust values, they are normalized. We may define normalized local trust value, $c(a_i, a_j)$, as follows:

$$c(a_i, a_j) = \frac{\max(s(a_i, a_j), 0)}{\sum_x \max(s(a_i, a_x), 0)} \quad (1)$$

This ensures that all values are between 0 and 1. These normalized local trust values are then aggregated.

Algorithm 1 Publish protocol for RDF data

```
for each RDF triple r do
  for each part (subject/predicate/object) r_p of r do
    attempt ← 0
    while true do
      id ← succ(h(r_p + attempt))
      ask a_id to store the triple r
      if a_id already at popularity threshold then
        a_id refuses to store r
        attempt ← attempt + 1
      else
        a_id stores r
        break
      end if
    end while
  end for
end for
```

Algorithm 2 Download protocol for RDF data

```
for each sub-query q in query Q do
  for each part (subject/predicate/object) q_p in q do
    attempt ← 0
    while true do
      id ← succ(h(q_p + attempt))
      Request triples D from a_id satisfying q_p
      if |D| < agent a_id's popularity threshold then
        break // the search for q_p is finished
      else
        attempt ← attempt + 1
      end if
    end while
  end for
end for
Download triples from servers in list obtained above
Locally compute query result from retrieved data
```

To keep the algorithm scalable and robust, eigenvector v is computed in a distributed and redundant fashion, where k different agents (score-managers) are responsible for computing each element of v. This conforms to Secure EigenTrust [20], except with global trust labels extended to objects as well as agents, and score-manager replicas grouped into Penny neighborhoods for better performance rather than disbursed throughout the ring.

3.2.8 Data Selection Procedure

Every object request (whether a traditional file download or RDF query) delivers to requesting agent a_{req} a set S of agents who can supply the object. If some respondents are malicious, some of these responses may differ. Agent a_{req} must choose among them based on their reputations. To do so, it partitions S by response. Let R denote the resulting equivalence relation, so that quotient set S/R is the set of agent groups, each of which returned a common response. For each partition $P \in S/R$ we compute the following evaluation function:

Algorithm 3 Data server selection procedure

```
if all members of S have trust 0 then
  select one server from S randomly
else if transaction is a police transaction then
  w_1 ← 0
  w_2 ← 0
  for each partition P ∈ S/R do
    choose partition P with probability f(P)
  end for
else
  w_1 ← 0.2
  w_2 ← 0.8
  B ← arg max_{P∈S/R} f(P)
  B ← arg max_{P∈B} |P|
  choose a partition randomly from set B
end if
```

$$f(P) = w_1 \frac{|P|}{|S|} + w_2 \frac{\sum_{a \in P} t(a)}{\sum_{a \in S} t(a)} + (1 - w_1 - w_2) \frac{1}{|S/R|} \quad (2)$$

where w_1 and w_2 are weights in $[0, 1]$ that prioritize each partition's relative size and reputation, respectively, in the evaluation. We determine acceptable values for w_1 and w_2 experimentally in §4.

Equation 2 is used by Algorithm 3 to resolve the selection choice. In the algorithm, *police transactions* are non-user transactions submitted by the security system during idle times in order to improve convergence. These are discussed in the next section.

4 Implementation and Results of Experimental Evaluation

We developed a Java implementation of Penny and tested its ability to weather several simulated attack scenarios. Efficiency and robustness of the network was evaluated in terms of the percentage of successful query responses. All experiments employ 20% malicious nodes and 10 pre-trusted agents [20, §4.5]. Throughout the simulation, we use SHA-256 for all hashing and 2^{160} for the identifier space size. We do not simulate the details of the underlying network and encryption operations, or anonymizing tunnels, since these are covered by prior works (see §2). Experiments are conducted under dynamic conditions, including peer joins and leaves, and neighborhood splitting and merging.

In our implementation and analysis of Penny, we focus on four classes of attacks:

- A malicious agent or collective might spread corrupt or incorrect data. For example, the malicious agent or collective might spread malicious code or circulate false facts.
- A malicious agent or collective might attach incorrect security labels to data. In particular, low-integrity

data might be ascribed a high-integrity label, or high-confidentiality data might be ascribed a low confidentiality label.

- A malicious agent or collective might attempt to learn which agents own certain data, perhaps as a prelude to staging additional attacks against those agents.
- A malicious agent or collective might attempt to generate a list of all data served by a particular agent, violating that agent'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 [18].

4.1 Bounding Neighborhood Size

As discussed in §3.2.1, the upper bound c for the size of a neighborhood must be chosen so as to balance high security (high c) with good performance (low c). We empirically determine a suitable value for c as follows. Each experiment consists of three phases of dynamic activity: (1) 1000 agents join the network, (2) 1000 random joins and leaves occur, and (3) 2000 more joins and leaves occur. All other network activities, including neighborhood splitting and merging, agent finger table updates, periodic EigenTrust runs, file downloads, etc., all occur randomly within all phases. The first two phases serve to initialize and stabilize the network; statistical results are gathered and reported only for phase 3. We tested networks with replication factors k ranging from 3 to 90, with 10 trials per replication factor. We also tested neighborhood size bounds of $c = 2k$ and $c = 3k$, obtaining the best results for $c = 3k$.

Figures 6(a) and 6(c) show that the number of neighborhood split and merge operations is greatly reduced when c is increased from $2k$ to $3k$. This is because splitting a size-$2k$ neighborhood results in one of size k, which is near the lower bound on neighborhood size. The new neighborhoods are therefore susceptible to merging, leading to oscillations between sizes k and $2k$, and many expensive merge/split operations. Choosing $c = 3k$ resolves this problem. Splitting size-$3k$ neighborhoods yields size-$\frac{3}{2}k$ neighborhoods, which must undergo considerable churn before they must be merged (at size k) or split again (at size $3k$). Even though the neighborhoods are larger, the vastly reduced number of split/merge operations leads to significantly fewer maintenance messages total, as shown in Figures 6(b) and 6(d). The curve in Figure 6(d) is smoother than the one in Figure 6(b) because of the elimination of the oscillations.

4.2 Results for File Downloads

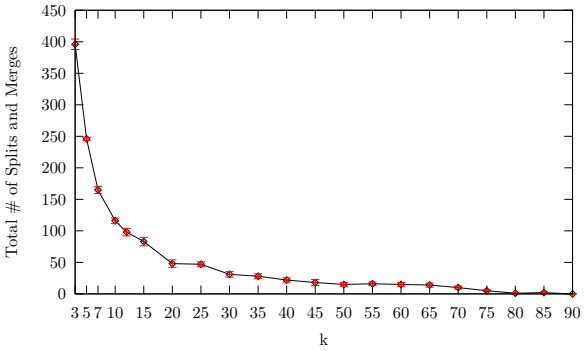
In this section we conduct different experiments for traditional file downloads in the presence of malicious agents, and show the robustness of Penny against these attacks. We simulate the publish protocol (Figure 4) and the request protocol (Figure 5). For these experiments, there are 1000 agents and 100 file objects in the system, and $k = 5$. We run 1000 downloads in the simulation, each of which uses the selection procedure in Algorithm 3.

Algorithm 3 makes the natural choice of preferring high- over low-reputation agents for user-submitted requests. We discovered that this tends to cause EigenTrust (and other reputation-based trust management systems) to converge slowly because low-reputation agents are so rarely exercised. To correct this, we introduced a new form of transaction, called a *police transaction*, that is designed to harmlessly exercise the system during idle periods rather than yield a correct result. Such transactions utilize low-reputation agents, providing higher-reputation agents additional opportunities to evaluate their answers. In our simulations, we used 50% police transactions.

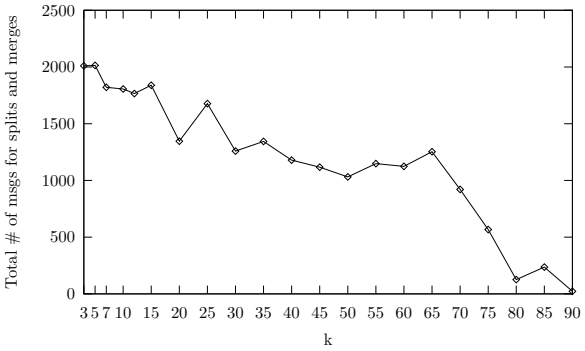
For non-police transactions, we placed greatest weight on reputations ($w_2 = 0.8$) and the remaining weight on consensus size ($w_1 = 0.2$). We consider each 20 downloads as one frame and thus show the frame position over time with 1000 downloads. After each frame, we run the EigenTrust algorithm and compute global trust values accordingly. For each type of experiment, we run it 5 times and take the average success rate. For all experiments, we pessimistically assume that all malicious agents know the identities of all the pre-trusted agents, and that they must display high trust for those agents in order to avoid lowering their own reputations. Thus, malicious agents trust only other malicious agents and pre-trusted agents.

In our *negative feedback experiment*, malicious agents always serve malicious files, and non-malicious agents who download the files always submit negative feedback for the transaction. Figure 7 shows that under these conditions, malicious agents fail to accrue high trust. Figure 7(a) is for a static network with no leaves or joins, and Figure 7(b) is for a dynamic network undergoing constant churn. As expected, convergence is slower in the presence of dynamic activity; the static network converges at about frame 10, whereas the dynamic doesn't until about frame 20. For both, we get a very high average success rate: 95.58% for the static network and 92.22% for the dynamic one, even with 20% malicious agents.

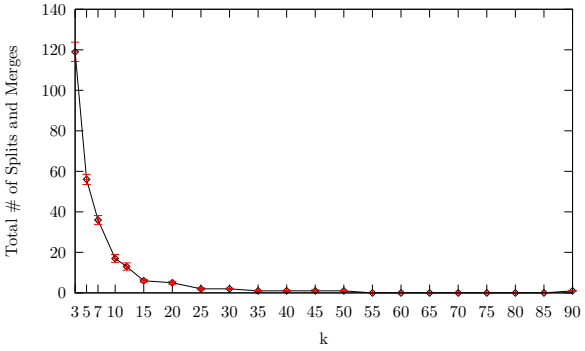
Figure 8 records the results of our *half-correct behavior experiment*, in which malicious agents provide correct files 50% of the time. Non-malicious agents always provide positive feedback for correct files and negative feedback for corrupt ones. Both static and dynamic networks converge quickly—at approximately frames 14 and 24, respectively. Average success rates were also still very high: 96.72% for the static network and 94.50% for the dynamic one. We further observe that the success rates are higher than each corresponding negative feedback experiment, since malicious agents provide correct files 50% of the time. On the other hand, convergence is slower because non-malicious agents take longer to identify the malicious agents.



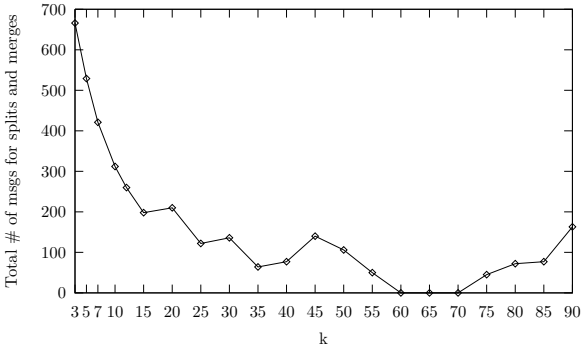
(a) splits and merges ($c = 2k$)



(b) split/merge messages ($c = 2k$)

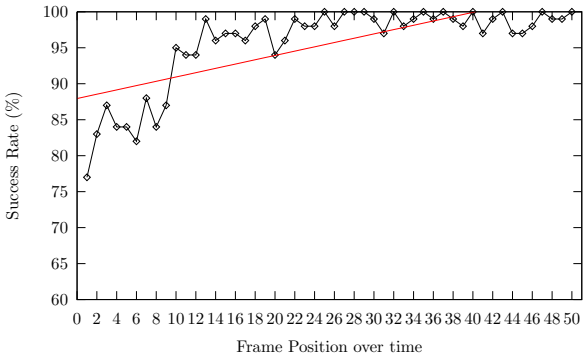


(c) splits and merges ($c = 3k$)

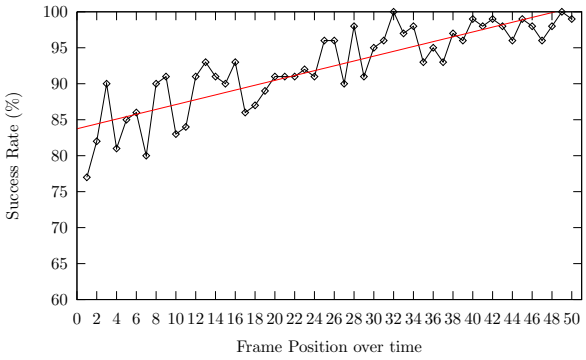


(d) split/merge messages ($c = 3k$)

Figure 6: Performance comparisons for $c = 2k$ and $c = 3k$

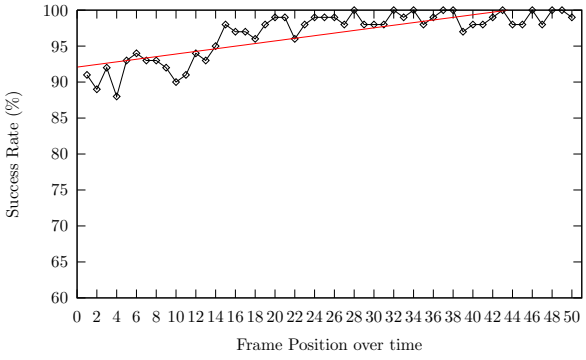


(a) static network

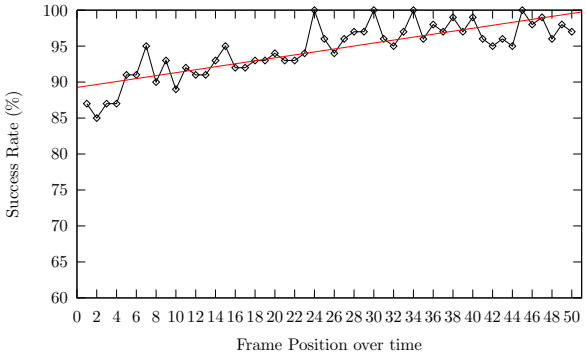


(b) dynamic network

Figure 7: Negative feedback experiment success rates

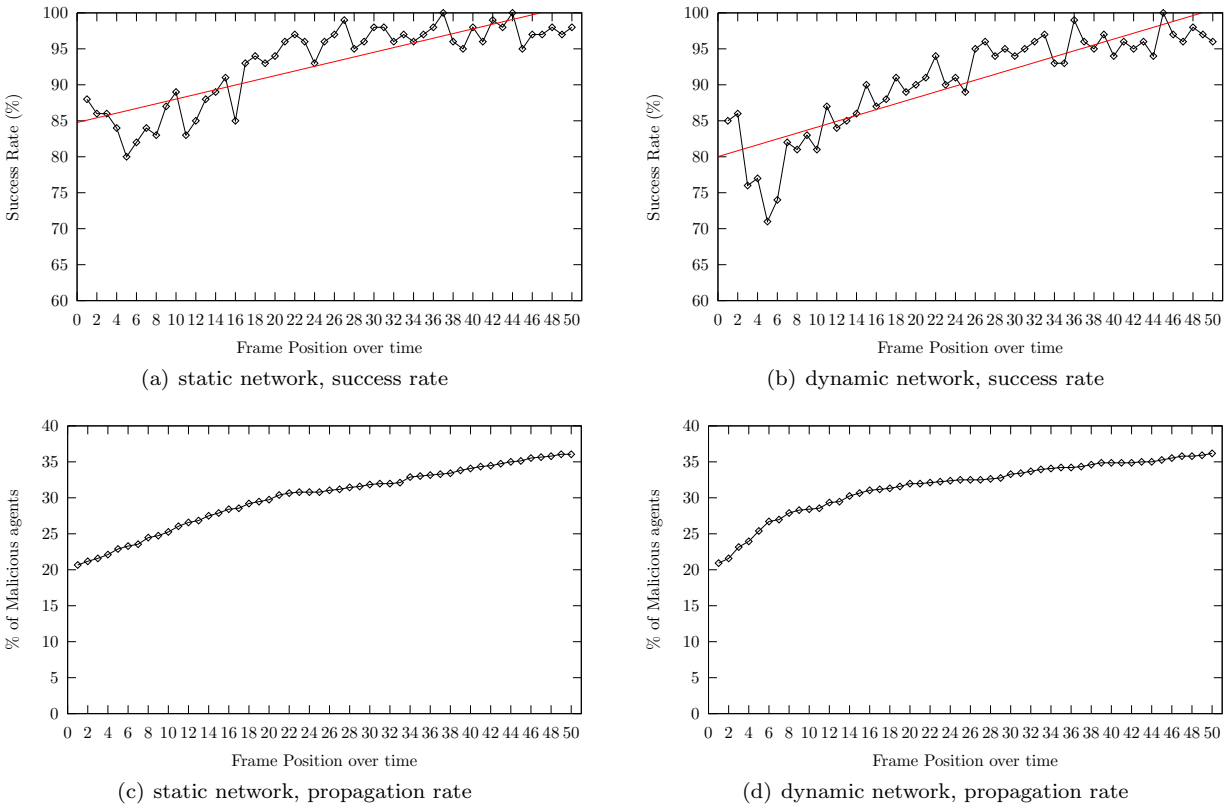


(a) static network



(b) dynamic network

Figure 8: Half-correct behavior experiment success rates



(a) static network, success rate

(b) dynamic network, success rate

(c) static network, propagation rate

(d) dynamic network, propagation rate

Figure 9: Malware propagation experiment success and propagation rates

Our *malware propagation* experiment next considers the pervasive problem of botnet malware infections of P2P file-sharing networks. In this experiment, non-malicious downloaders of malicious files have a 20% chance of becoming infected and exhibiting malicious behavior thereafter. Malicious agents behave the same as in the half-correct behavior experiment. In both static (Figure 9(a)) and dynamic (Figure 9(b)) networks, success rates initially drop as previously high-reputation agents suddenly attack the system. However, the reputation system adapts and around frame 16 the non-malicious agents manage to largely isolate the infection. The count of malicious agents continues to grow monotonically, as seen in Figures 9(c) and 9(d), because the experiment includes no facility for disinfection. But the growth slows, and any new malicious agents are identified relatively quickly by the non-malicious majority. The average success rates were 93.04% for static networks and 90.44% for dynamic ones.

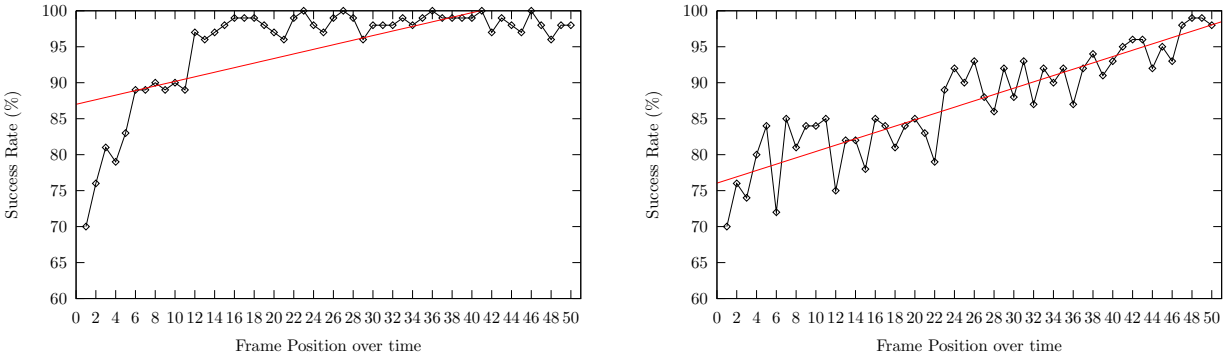
4.3 Results for RDF Datasets

We next present experiments for RDF dataset downloads in the presence of malicious agents, and show the robustness of Penny networks. We simulate the publish protocol (Algorithm 1) and download protocol (Algorithm 2). The rest of the experimental setup is same as in §4.2. We use the LUBM100 [16] dataset for our experiments, which is broadly used by researchers for similar evaluations [15]. The LUBM data generator yields datasets in RDF/XML format, which we converted to N-triples format. For download or query purposes, we use atomic triple queries and conjunctive multi-predicate queries (cf., [8]). We conduct the same three sets of experiments for RDF datasets as reported in §4.2.

For the negative feedback experiment (Figure 10) we see average success rates of 95.12% for static networks and 87.26% for dynamic ones. These are slightly lower than the corresponding rates for non-RDF file downloads because of the additional number of transactions required to successfully answer RDF queries. If any sub-query fails, the entire query fails. In addition, the coalesced chaining implemented by Algorithm 2 requires additional transactions to retrieve popular triples. Convergence rates are slightly lower for the same reason. Despite this, both success rates and convergence rates remain quite high for a network with so much malicious population.

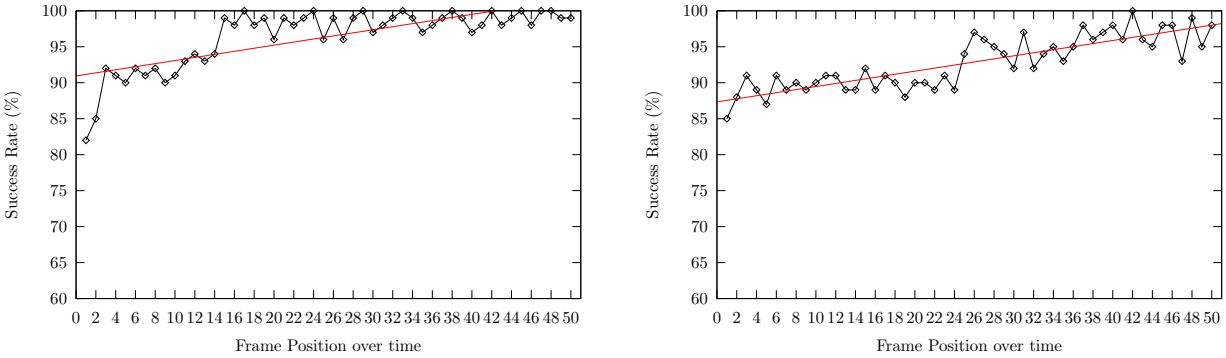
The half-correct behavior experiment exhibits even faster convergence, as seen in Figure 11. The static network converges at about frame 15, and the dynamic at 25. Average success rates were similarly high at 96.46% and 92.78%, respectively.

While malware is not possible in RDF data to our knowledge, for the sake of completeness we replicated the malware propagation experiment for the RDF publish and download protocol. Results are reported in Figure 12. Both static and dynamic networks exhibited fast convergence; about frame 19 for the static network and 29 for the dynamic one. Success rates were similarly promising, being 92.90% and 88.98% on average for the static and



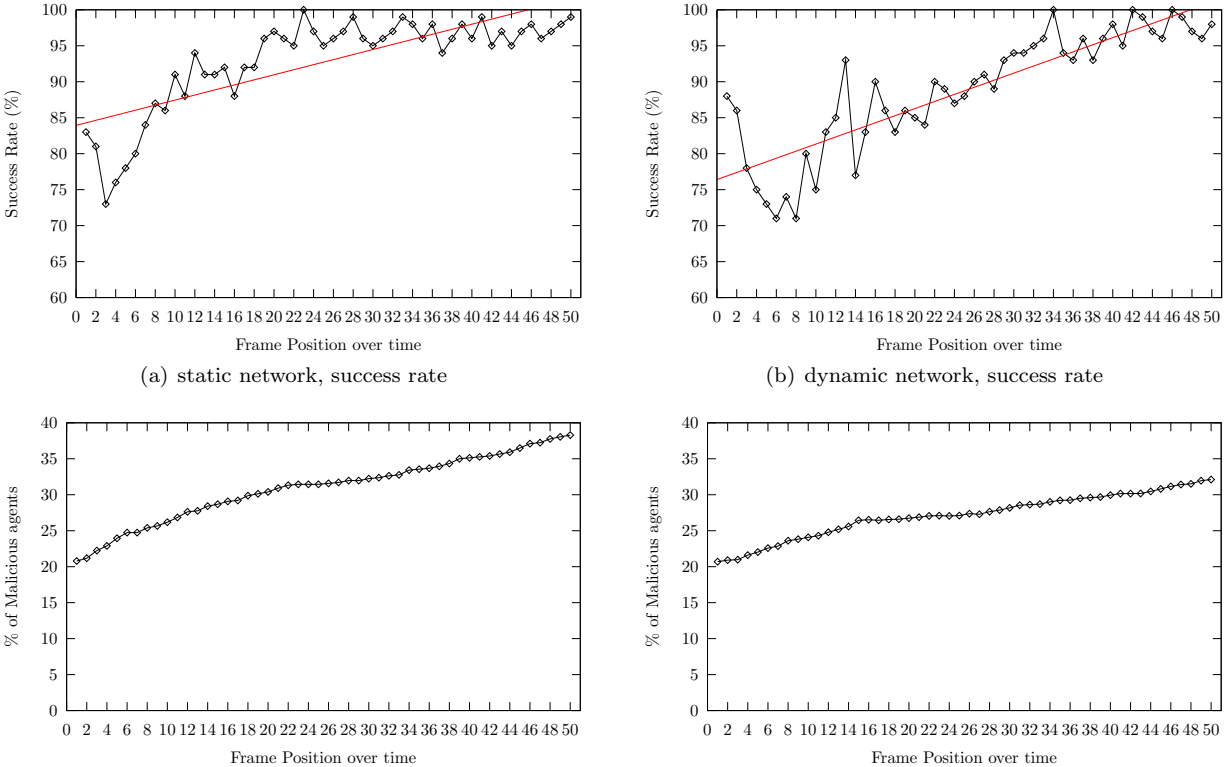
(a) static network (b) dynamic network

Figure 10: RDF negative feedback experiment success rates



(a) static network (b) dynamic network

Figure 11: RDF half-correct behavior experiment success rates



(a) static network, success rate (b) dynamic network, success rate

(c) static network, propagation rate (d) dynamic network, propagation rate

Figure 12: RDF malware propagation experiment success and propagation rates

dynamic cases, respectively. Again, these are slightly lower than for file downloads because of the higher complexity of the RDF protocol. As before, both networks exhibit an initial drop in success but manage to adapt and recover fairly smoothly.

5 Discussion

The high success rates and strong convergence properties experimentally observed in §4 can be traced largely to Penny's support for exceptionally high data replication via its neighborhood topology. Label retrieval is efficient in Penny, requiring approximately the same number of messages as object lookup in a Chord network, but with k independent replicas of each label. An agent can retrieve any object's global 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, from which one response is selected via Algorithm 3.

Penny inhibits the spread of low-integrity data (e.g., malware) by maintaining a global integrity label for each object shared over the network. Agents wishing to avoid such data can therefore consult each object's global 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.

In addition to global integrity labels, Penny also maintains global confidentiality labels for objects. Agents can use these labels as a basis for selectively serving data to other agents—possibly based on the requester's trust level, global confidentiality label, or other credentials.

An object's global security labels are determined by the votes of other agents in the network via EigenTrust [20]. 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 other voters. Penny uses EigenTrust to track agent reputations and to prevent malicious agents from accruing good reputations.

Secure hashing and replication are both employed to protect against malicious key-holders and score-managers who might falsify an object's global integrity labels or an agent's global 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 $\lfloor b/2 \rfloor$ agents in a neighborhood must be malicious in order to subvert a reputation, where $b \geq k$ is the neighborhood size.

Malicious peers cannot elevate their own reputations by switching IP addresses or creating false network accounts because all agent and object reputations start at zero in Penny (cf., [20]). An agent or object acquires a positive reputation only by participating in positive transactions with other agents. Agents with established reputations then report positive feedback for those transactions, elevating the new agent's reputation.

Unlike Penny, Chord [36] 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 (see Figure 5). 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 (assuming the collective is small relative to the size of the network). Thus, Penny enforces a notion of object ownership privacy.

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 whether 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 whether 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. So a large identifier space provides natural resistance to guessing attacks.

6 Conclusion and Future Work

Penny efficiently supports global trust labels, data integrity labels, and data confidentiality labels in a fully decentralized, structured, peer-to-peer network. Global labeling assures convergence for all security queries, while decentralization avoids centralized points of failure typically associated with centralized label servers. Its reputation management system applies and extends EigenTrust [20], distributed hash tabling based on Chord [36], and anonymizing tunnels based on Tarzan [12, 13] or SurePath [43]. The security labeling scheme preserves the

efficiency of network operations; lookup cost including label retrieval is $O(\log N + k)$, where N is the network size and k is a constant replication factor.

We developed a Penny client in Java and tested it under eight attack simulations. The results illustrate Penny's efficiency and reliability over realistic network operations, including high dynamic churn; object publications, lookups, and downloads; and regular reputation maintenance via the Secure EigenTrust algorithm. The results also demonstrate the robustness of Penny in the presence of malicious agents. We obtain extremely high average success rates for all experiments even when 20% of the network is malicious. Experiments show that success rates remain high even with relatively complex publish protocols, such as those used to manage RDF data.

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 accomplishes this by decoupling object-owner information through a cryptographically protected layer of indirection.

Our implementation and 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 (cf., [18]). We intend to consider such attacks in future work.

Future research should also consider how to enforce information flow 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 data items 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, but it is unclear how to ensure that such violations get reported (since typically the only witnesses are the malicious agents involved in leaking the data). Enforcing strong confidentiality policies in P2P networks therefore remains an interesting open problem.

Acknowledgment

This material is based upon work supported in part by the U.S. National Science Foundation (NSF) under grant #0959096. Any opinions, findings, conclusions, or recommendations expressed are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *Proc. 10th ACM Int. Conf. Information and Knowledge Management (CIKM)*, pages 310–317, 2001.

[2] Y. Bachrach, A. Parnes, A.D. Procaccia, and J.S. Rosenschein. Gossip-based aggregation of trust in decentralized reputation systems. *J. Autonomous Agents and Multiagent Systems (AAMAS)*, 19(2):153–172, 2009.

[3] A. Berns and E. Jung. Searching for malware in BitTorrent. Technical Report UICS-08-05, University of Iowa, Department of Computer Science, 2008.

[4] BitTorrent. http://www.bittorrent.com, 2012.

[5] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance checking in the PolicyMaker trust management system. In *Proc. 2nd Int. Financial Cryptography Conf. (FC)*, pages 254–274, 1998.

[6] N. Borisov. *Anonymous Routing in Structured Peer-to-Peer Overlays*. PhD thesis, The University of California at Berkeley, 2005.

[7] M. Cai, M. Frank, J. Chen, and P. Szekely. MAAN: A multi-attribute addressable network for grid information services. In *Proc. 4th Int. Workshop Grid Computing*, pages 184–191, 2003.

[8] M. Cai, M.R. Frank, B. Yan, and R.M. MacGregor. A subscribable peer-to-peer RDF repository for distributed metadata management. *J. Web Semantics*, 2(2):109–130, 2004.

[9] G. Ciaccio. Improving sender anonymity in a structured overlay with imprecise routing. In *Proc. 6th Int. Workshop Privacy Enhancing Technologies (PET)*, pages 190–207, 2006.

[10] F. Cornelli, E. Damiani, S.D.C. di Vimercati, S. Paraboschi, and P. Samarati. Choosing reputable servents in a P2P network. In *Proc. 11th Int. Conf. World Wide Web (WWW)*, pages 376–386, 2002.

[11] E. Damiani, S.D.C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *Proc. 9th ACM Conf. Computer and Communications Security (CCS)*, pages 207–216, 2002.

[12] M.J. Freedman, E. Sit, J. Cates, and R. Morris. Introducing Tarzan, a peer-to-peer anonymizing network layer. In *Proc. 1st Int. Conf. Peer-to-peer Systems (IPTPS)*, pages 121–129, 2002.

[13] M.J. Freedman, E. Sit, J. Cates, and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proc. 9th ACM Conf. Computer and Communications Security (CCS)*, pages 193–206, 2002.

[14] Gnutella. http://www.gnutella.com, 2010.

[15] Y. Guo, Z. Pan, and J. Heflin. An evaluation of knowledge base systems for large OWL datasets. In *Proc. 3rd Int. Semantic Web Conf. (ISWC)*, pages 274–288, 2004.

[16] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Semantics*, 3(2-3):158–182, 2005.

[17] M. Gupta, P. Judge, and M.H. Ammar. A reputation system for peer-to-peer networks. In *Proc. 13th ACM Int. Workshop Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 144–152, 2003.

[18] K.W. Hamlen and W. Hamlen. An economic perspective of message-dropping attacks in peer-to-peer overlays. *Intelligence and Security Informatics (ISI)*, 1(6), 2012.

[19] Q. He, J. Yan, H. Jin, and Y. Yang. ServiceTrust: Supporting reputation-oriented service selection. In *Proc. 7th Int. Joint Conf. Service-Oriented Computing (ICSOC-ServiceWave)*, pages 269–284, 2009.

[20] S.D. Kamvar, M.T. Schlosser, and H. Garcia-Molina. The EigenTrust algorithm for reputation management in P2P networks. In *Proc. 12th Int. Conf. World Wide Web (WWW)*, pages 640–651, 2003.

[21] KaZaA. http://www.kazaa.com, 2012.

[22] A. Khaled, M.F. Husain, L. Khan, K.W. Hamlen, and B. Thuraisingham. A token-based access control system for RDF data in the clouds. In *Proc. 2nd IEEE Int. Conf. Cloud Computing Technology and Science (CloudCom)*, pages 104–111, 2010.

[23] S. Lee, R. Sherwood, and B. Bhattacharjee. Cooperative peer groups in NICE. In *Proc. 22nd Annual Joint Conf. IEEE Computer and Communications Society (INFOCOM)*, pages 1272–1282, 2003.

[24] E. Liarou, S. Idreos, and M. Koubarakis. Publish/subscribe with RDF data over large structured overlay networks. In *Proc. Int. Conf. Databases, Information Systems, and Peer-to-peer Computing (DBISP2P)*, pages 135–146, 2005/2006.

[25] E. Liarou, S. Idreos, and M. Koubarakis. Continuous RDF query processing over DHTs. In *Proc. 6th Int. The Semantic Web Conf. (ISWC) and 2nd Asian Semantic Web Conf. (ASWC)*, pages 324–339, 2007.

[26] F. Marozzo, D. Talia, and P. Trunfio. A peer-to-peer framework for supporting MapReduce applications in dynamic cloud environments. In Nick Antonopoulos and Lee Gillam, editors, *Cloud Computing: Principles, Systems and Applications*, pages 113–125. Springer, 2010.

[27] S. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, University of Stirling, 1994.

[28] Napster. http://www.napster.com, 2012.

[29] A. Newman, J. Hunter, Y.-F. Li, C. Bouton, and M. Davis. A scale-out RDF molecule store for distributed processing of biomedical data. In *Proc. Semantic Web for Health Care and Life Sciences Workshop (HCLS) at the 17th Int. World Wide Web Conf. (WWW)*, 2008.

[30] A. Newman, Y.-F. Li, and J. Hunter. A scale-out RDF molecule store for improved co-identification, querying and inferencing. In *Proc. 4th Int. Workshop Scalable Semantic Web Knowledge Base Systems (SSWS) at the 7th Int. Semantic Web Conf. (ISWC)*, 2008.

[31] J.M. Pujol, R. Sangüesa, and J. Delgado. Extracting reputation in multi-agent systems by means of social network topology. In *Proc. 1st ACM Int. Joint Conf. Autonomous Agents and Multiagent Systems (AAMAS)*, pages 467–474, 2002.

[32] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable, content-addressable network. In *Proc. ACM Conf. Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 161–172, 2001.

[33] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large scale peer-to-peer systems. In *Proc. IFIP/ACM Int. Conf. Distributed System Platforms (Middleware)*, pages 329–350, 2001.

[34] J. Sabater and C. Sierra. Reputation and social network analysis in multi-agent systems. In *Proc. 1st ACM Int. Joint Conf. Autonomous Agents and Multiagent Systems (AAMAS)*, pages 475–482, 2002.

[35] H. Schulze and K. Mochalski. Internet study 2008/2009. Technical report, ipoque, 2009. http://www.ipoque.com/sites/default/files/mediafiles/documents/internet-study-2008-2009.pdf.

[36] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM Conf. Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 149–160, 2001.

[37] N. Tsybulnik, K.W. Hamlen, and B. Thuraisingham. Centralized security labels in decentralized p2p networks. In *Proc. Annual Computer Security Applications Conf. (ACSAC)*, pages 315–324, 2007.

[38] W.J.A. Verheijen. Efficient query processing in distributed RDF databases. Master's thesis, Technische Universiteit Eindhoven, 2008.

[39] F. Wang, Y. Zhang, and J. Ma. Modelling and analyzing passive worms over unstructured peer-to-peer networks. *Int. J. of Network Security (IJNS)*, 11(1):39–45, 2010.

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

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

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

[43] Y. Zhu and Y. Hu. SurePath: An approach to resilient anonymous routing. *Int. J. Network Security (IJNS)*, 6(2):201–210, 2008.

Safwan Mahmud Khan is a computer science Ph.D. student in the Software Security Lab at The University of Texas at Dallas working under the supervision of Professor Kevin Hamlen. His main research area has been cloud computing security and its extension to secure peer-to-peer systems. Before coming to UT Dallas, Safwan obtained an M.S. degree in Computer Science from The University of Texas at Arlington. He worked there in the Information Security (iSec) Lab with his advisor Professor Matthew Wright. There his research area was related to anonymity in large scale peer-to-peer systems. He earned his B.Sc. in Computer Science and Engineering from the University of Dhaka in Bangladesh. Safwan has seven international conference papers and two journal papers. He achieved 4th position in ACM ICPC (international programming contest), 2002, Dhaka, Bangladesh site. Besides his academic research experience, Safwan has more than three years of industry experience. His research interests include security in cloud computing, secure peer-to-peer systems, network security and privacy, data mining, and algorithms.

Dr. Kevin W. Hamlen is an Associate Professor of Computer Science at The University of Texas at Dallas. He holds Ph.D. and M.S. degrees in Computer Science from Cornell University, and a B.S. degree in Computer Science and Mathematical Sciences from Carnegie Mellon University. His research concerns language-based approaches to software security, malware attacks and defenses, and distributed computing security, including clouds and peer-to-peer networks. He is the winner of numerous research awards, including a Young Investigator Program (YIP) award from the U.S. Air Force Office of Scientific Research and a CAREER award from the U.S. National Science Foundation. He is the author of over forty peer-reviewed conference and journal articles on computer security.