Scrivener: Providing Incentives in Cooperative Content Distribution Systems[*]

Animesh Nandi[1], Tsuen-Wan "Johnny" Ngan[1], Atul Singh[1], Peter Druschel[2], and Dan S. Wallach[1]

[1]Department of Computer Science, Rice University.
[2]Max Planck Institute for Software Systems.

Abstract. Cooperative peer-to-peer (p2p) applications are designed to share the resources of participating computers for the common good of all users. However, users do not necessarily have an incentive to donate resources to the system if they can use the system's services for free. In this paper, we describe Scrivener, a fully decentralized system that ensures fair sharing of bandwidth in cooperative content distribution networks. We show how participating nodes, tracking only first-hand observed behavior of their peers, can detect when their peers are behaving selfishly and refuse to provide them service. Simulation results show that our mechanisms effectively limit the quality of service received by a user to a level that is proportional to the amount of resources contributed by that user, while incurring modest overhead.

1 Introduction

This paper concerns itself with the fair sharing of resources in cooperative peer-to-peer (p2p) systems. In such a system, participating nodes are expected to contribute a fraction of their resources in exchange for access to a service provided by the system. Clearly, if participants fail to contribute enough resources to offset the load imposed by all users, then the system's stability and usability may be in danger.

Experience with file-sharing systems like Gnutella and KaZaA shows that many users may choose to consume the system's services without providing any of their own resources for the use of others [2]. The problem is that participants have no natural *incentive* to provide services to their peers if it is not somehow required of them. Users more closely resemble economically "rational" agents who are willing to follow the protocol only if that behavior maximizes the node's "utility" from the p2p network. If there is no immediate penalty for selfish behavior, then nodes will behave selfishly, and the p2p system will fail. Economic theory calls these users "free riders" or "freeloaders," and the resulting scenario "the tragedy of the commons" [21].

Ideally, we would like to design a system where nodes, acting in their own best interest, behave collectively to maximize the common welfare. Designing such a system without a centralized authority that has complete knowledge of the system becomes a distributed algorithmic mechanism design (DAMD) problem [12]. DAMD is a current area of study that combines computational tractability in theoretical computer science with incentive-compatible mechanism design in the economics literature. It provides a useful framework for considering p2p systems [27, 28, 33]. This paper considers

incentives-based mechanisms that ensure fair sharing, focusing on cooperative systems where network bandwidth is the contented resource.

One way to enforce fairness is to have, for each node in the system, a set of other nodes account for that node's actions and approve requests according to the system's policy. KARMA [36] is an example of such a system. However, coordinating the actions of this auditor set requires both cryptographic operations and additional communication *every* time a peer issues or responds to a request. This can add substantial overhead and latency to the system. Moreover, this approach introduces the additional problem of how to incentivize the auditor set to perform its function correctly [36].

Instead, we hypothesize that a normal p2p node, monitoring the behavior of its overlay neighbors, will have sufficient information to locally identify and discourage selfish behavior. When nodes give preferential service to peers who follow the rules, rational agents will choose to follow the rules to receive better services. An early example of a p2p system built in this fashion is BitTorrent [7], where nodes employ a "tit-for-tat" policy, preferring to transmit content to other nodes who are willing to return the favor. BitTorrent focuses on the case where all peers are interested in the same content, e.g., different blocks of a large software distribution. Thus, it is common that two peers simultaneously have a block that is of interest to the other, enabling a "clean swap."

In this paper, we are attempting to solve the more general problem of a content distribution system where peers are interested in obtaining objects from a large collection, consisting of both popular and unpopular objects. In this setting, a simultaneous swap of content is rarely possible. Instead, it is necessary to maintain a history of interactions (in terms of credit and debt) with a peer to make decisions concerning the peer in the future. Moreover, the good will accumulated by a BitTorrent node is lost when that node completes downloading the object and leaves the system. BitTorrent nodes have no incentive to stay around and help their peers. In our system, we wish to encourage such behavior by allowing peers to accumulate credit that can be redeemed at a later time, for possibly unrelated content.

The remainder of this paper is structured as follows. Section 2 describes the model and the goals of our system. In Section 3, we present the design of Scrivener, a system that enforces fair bandwidth sharing in a cooperative content distribution system. Section 4 describes the implementation of Scrivener in the context of an existing content distribution system. We present simulation results in Section 5. Finally, Section 6 discusses related work and Section 7 concludes.

2 System model and goals

We consider cooperative content distribution systems where participants wish to obtain content stored on other participants' computers. Content is assumed to be published by its owner and disseminated into the system for distribution. We assume that, at least for popular objects, the owner has insufficient bandwidth to service every possible request and wishes to leverage the bandwidth available among other nodes in the system.

The set of participating nodes is assumed to form an overlay network. Scrivener is based on mechanisms that in principle can be applied to both unstructured [17, 23] and structured overlay networks [30, 34], as long as they meet the following minimal requirements: (1) Each node in the overlay communicates directly with only a bounded (i.e., constant or logarithmic in the size of the overlay) number of overlay *neighbors*; (2) the overlay has a mechanism to discover new overlay neighbors; and, (3) the overlay supports a search primitive that discovers, when given a valid content identifier, one or more overlay paths to a node that stores content associated with that identifier.

We further assume that node identifiers cannot be created and discarded freely. The mechanisms we will describe are all based on observing which nodes have behaved properly and which have not. If nodes could misbehave under one identity, only to discard it and assume another identity, then there would be no incentive for proper behavior. Such "Sybil attacks" [11] are a fundamental issue in overlay networks and a host of different attacks become possible unless nodeIds are somehow controlled. For the purposes of our research, we require an external solution to Sybil attacks. For example, Castro et al. [6] address this by requiring a trusted authority to issue certificates that bind a nodeId to a public key; they also describe a weaker, decentralized approach to issuing such certificates. Since we are primarily interested in supporting systems for the distribution of legal content, maintaining user anonymity is not a design goal of Scrivener. If, however, an anonymity-preserving defense against Sybil attacks was available, Scrivener might still be applicable.

2.1 Attack model

The adversarial model assumed by Scrivener is limited to simple *freeloading* behavior, whose only objective is to obtain service without contributing an equivalent fair share of bandwidth to the system. This is in contrast to more general *malicious* behavior, where the objective of the attacker may include obtaining unauthorized access to content, corrupting or censoring content, or denying or degrading service to other users. Mechanisms to prevent or mitigate such behavior (e.g., sealed and self-certifying content [15], content entanglement [37], Castro et al. [6]'s secure routing primitive) may be employed to complement Scrivener. Most p2p systems are already engineered to be robust against traffic loss due to network failures. In the extreme case of a node refusing to properly forward low-level traffic, that nodes' neighbors could flag the node as unresponsive and would likely remove the node from the network. As such, we are primarily concerned with *application layer* freeloading, where the application's goal is the sharing and distribution of content of varying size and popularity.

It is useful to consider freeloading separately from more general malicious behavior, particularly when in many systems it is much easier to freeload than to mount a malicious attack. In KaZaA [23], for example, a client configured to have minimum upload bandwidth and turning off the super-peer flag suffices to freeload. A malicious attack, on the other hand, would require considerable technical expertise. Thus, the fraction of users who have the motivation and ability to freeload is likely to far exceed the fraction of users that are intent and able to mount a malicious attack.

Accordingly, the two threats call for different mechanisms. A defense against freeloading must be effective and efficient even when a large fraction of participants attempt to freeload. A defense against malicious behavior can, and often must, assume that malicious behavior is limited to a small minority of users. We expect that a production content distribution system would include both types of mechanisms. For the remainder of this paper, we will focus exclusively on detecting and preventing freeloading.

2.2 Goals

Scrivener's goal is to achieve fair sharing of bandwidth in content distribution systems. The key aspects of this goal are summarized below.

– *Fairness*. The system must ensure that participants receive a quality of service that is proportional to the amount of bandwidth they are actually contributing to the system. Furthermore, no participant should be permitted to perpetually consume

resources in excess of their contributions at the expense of another participant. This provides an incentive for nodes not to freeload.

– *Low overhead.* The overhead imposed by the mechanisms used should be modest. Moreover, the marginal cost related to ensuring fairness when downloading an object should be low, to ensure efficiency despite small object sizes.
– *Robustness.* The system should retain the above properties even in the presence of large numbers of freeloaders and in the presence of modest churn.

3 Design

Fundamentally, Scrivener is based on the idea of a pairwise exchange of content between overlay participants. This is similar in spirit to BitTorrent, where participants exchange content fragments "tit-for-tat." However, unlike BitTorrent, Scrivener considers the general case of a content distribution system where participants with different interests choose from a large set of content objects. In such a system, it is unlikely that two overlay neighbors are simultaneously interested in each other's content, which would enable a "clean swap." Making pairwise exchange work in a general content distribution network presents several challenges. The basic concepts of Scrivener include:

Relationships: A Scrivener node maintains a relationship with each of its overlay neighbors. Each of the two nodes involved in a relationship maintains a credit and a confidence value for the other node, defined below. These values are maintained in persistent storage and are remembered even as a node departs and subsequently rejoins the overlay. The values are maintained and used only locally to a given node.

Credit: Credit is the difference between the amount of data sent to and the amount of data received from the peer.[1] Negative values of credit are called *debt*.

Confidence: The positive confidence value for the neighbor is calculated according to an additive increase, multiplicative decrease policy, based on the success or failure of content requests that were forwarded to the neighbor. The confidence value is used in deciding how to forward requests during content search and it is used to compute the credit limit (defined below) granted to the neighbor node.

Building on these core ideas, easily applicable to any p2p content distribution system, we can invent a number of mechanisms:

Maintaining credit / debt: To enable non-simultaneous pairwise swapping, each Scrivener node maintains a record of credit / debt with each of its overlay neighbors. We wish to enable a node *A* to obtain content from another node *B*, even when *A* may not currently have any content of interest to *B*. *A* can repay the resulting debt to *B* at a future time, when *B* happens to be interested in some content held by *A*. A node honors requests from a peer if and only if that peer is in good standing, i.e., the peer's debt is below a certain limit.

Limiting generosity: To bootstrap the system, one node must be willing to extend a loan to another node with which it has had no prior relationship. However, such loans must not enable freeloading. A Scrivener node *A* grants a small initial credit to each node *B* that *A* has chosen to initiate a relationship with. However, node *B* does not necessarily grant *A* any credit in return. As *A* and *B* interact and respond to each other's requests, the confidence among the peers, and thus the amount of credit granted, can increase over time.

---

[1] We assume here that the cost of transferring an object is equal to the size of the object in bytes. It is equally possible to define certain objects as more valuable than others.

Limiting relationships: Each node initiates relationships with only a limited number of peers, typically the neighbors chosen by the overlay network. This limits the amount of state maintained by each node and it limits the total credit a node grants its peers.

Transitive trading: What if a node wishes to obtain a content object not held by any of its overlay neighbors? We need a mechanism that allows a node to use the credit it has with its neighbors to obtain content from a more distant node that has the desired content, but with which it does not have a pre-existing relationship. Transitive trading is such a mechanism. Performing a transitive trade involves finding a path from the requester to a content holder such that each node along the path is in good standing with the subsequent node. Then, the content holder sends the content to the requester, and each node along the path credits the subsequent node.

3.1 Relationships

Each Scrivener node maintains relationships with a small number of other nodes, typically its overlay neighbors, as selected by the overlay protocol. More precisely, any two nodes in the overlay network form a relationship if and only if at least one of them has the other in its overlay neighbor table. A Scrivener node A grants a small initial confidence value (and thus a small credit limit) to any node that A has chosen as a neighbor, but it assigns an initial confidence of zero (and thus no credit) to any node that has invited A to be a neighbor. This prevents freeloaders from obtaining a large credit limit by initiating many relationships with many nodes, perhaps pretending that its normal neighbors have failed.[2]

The small initial credit limit allows neighbors chosen by A to request content from A, and it allows A to request content from legitimate nodes who have chosen A as a neighbor. As content is exchanged, the parties gain more confidence in each other and gradually grant each other larger credit limits. Our scheme puts newcomers at a disadvantage; they need to initiate relationships, forcing them to grant credit and offer service while receiving little in return initially. This is the price for defending against freeloaders in any reputation-based system. However, as we will show, the initial sacrifice is rewarded quickly as the node establishes confidence and gains credit with its neighbors.

When a Scrivener node A finds that one of its neighbors B has accumulated debt in excess of its credit limit, it ceases to accept requests from B. Regardless, A continues to make requests to B in order to give B the opportunity to pay back its debt. Likewise, A may find that the confidence value of one of its neighbors B goes to zero, perhaps because B has repeatedly failed to fulfill requests from A even though A is in good standing with B. In this case, A ceases to make requests via B or to accept requests from B. From A's perspective, B might as well not be a part of the overlay network. A then uses existing mechanisms provided by the overlay network to replace B with a different, and hopefully more cooperative, neighbor.

In principle, a Scrivener node must maintain a record of its past overlay neighbors indefinitely. Erasing a negative record would amount to forgiving debt, and would enable freeloading. In practice, it is acceptable to delete records of nodes that have been offline for long periods, perhaps a year, thus seriously inconveniencing freeloaders who

---

[2] Overlay network systems are generally engineered to assume a high rate of node failure and include elaborate mechanisms to locate previously unknown nodes and form new relationships in order to preserve important invariants, including the degree of node-to-node connectivity and of file replication. As a result, we need to limit the benefits automatically granted to a node solely because it happens to be a peer.

wish to exploit the resulting loophole. Storing a year's worth of records is reasonable as these records are very compact: only a nodeId and two integer values, the credit and confidence values, are required. Such concise records could easily scale to track the millions of neighbors that a node might see in a year's time.

Note also that due to the pairwise relationships, freeloader cannot benefit from collusion. While colluding freeloaders may be able to convince legitimate nodes to shift credit from one freeloader to another, the total credit will be unchanged.

3.2   Confidence

Scrivener nodes keep a confidence estimate for each of their overlay neighbors. The confidence value serves two purposes: (1) it determines the magnitude of the credit limit granted to a neighbor and (2) it can be used to bias overlay routing decisions towards cooperative neighbors.

The confidence assigned by a node to its neighbor is based on the history of their relationship. The confidence estimate has the following properties: (1) As nodes exchange content, the confidence increases slowly; (2) The confidence drops rapidly once a neighbor starts to misbehave; (3) The confidence is bounded to limit the damage caused by a node that plays by the rules for an extended period and then starts to freeload. An additive increase, multiplicative decrease (AIMD) strategy offers a simple implementation of these properties.

3.3   Transitive trade

In p2p content distribution systems with a large content set, the odds are small that a desired object can be found on an immediate overlay neighbor of the node wishing to fetch that object. We need a way for nodes to trade their credits and debts with one another, and we would like to avoid the overhead of digital cash or other cryptographic schemes. Instead, we designed an incremental trading strategy we call *transitive trade*, which works by identifying a *credit path* from a source node to a node that has the desired object. In a credit path, each node in the path either has credit with the next node, or its debt is below the next node's credit limit. We describe a scheme to locate such paths in Section 4.3.

Conceivably, once we have identified a credit path, we could rearrange all the credits in the path such that the destination node now owes something not to its predecessor in the route, but instead to the source of the route. This is illustrated in Figure 1. A series of debts, where B owes A, C owes B, and so forth until Z owes its predecessor could all be replaced with a direct debt from Z to A. Z can now cancel this debt by providing A with the desired content.

To make debt swapping work, we need a protocol that is robust against any node in the trading chain cheating. For example, a node could attempt to cancel a debt that it owes without giving up the debt owed to it by the successor in the trading chain. Rather than resorting to a complex cryptographic commitment protocol, we take a straightforward, incremental approach. The protocol is depicted in Figure 2.

1: Credit path discovery: A first routes a "path discovery" message (PD) towards Z. As a side effect, A "pays" B for this message, B pays C, and so forth until Z is paid. At the same time, each node reduces its confidence in its successor as if the request had failed (even though it may be working perfectly well). This design avoids the need to maintain timeouts to detect and react to failures. The credit path discovery might fail for a number of reasons, ranging from a freeloader dropping the message to network failures (see
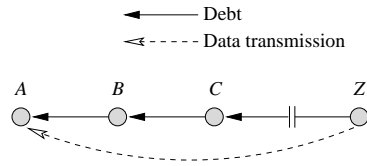
Fig. 1. Using a credit path to leverage a chain of credit to obtain content directly from a non-neighbor node.
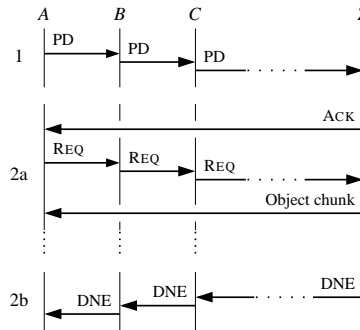

Fig. 2. The stages in the transitive trade protocol.

Section 4.3). The effect is that every node that forwarded the request will have reduced confidence in its successor. Furthermore, the last node in the chain effectively keeps the credit originally transfered from A.

2a: Object exists: Upon receiving the request, Z transmits a confirmation message (ACK) directly to A. A now routes a request message (REQ) for a chunk of the content object along the existing credit path, paying for the chunk as a side-effect of the message transmission. Z transmits the requested object chunk directly to A. A repeats this step until it has obtained the last chunk of the object. A final message, announcing A's success, causes each node to adjust the confidence value of its successor to compensate for the reduction in step (1), plus an additional confidence gained as a result of the trade.

2b: Object does not exist: Upon receiving the request, Z routes a "does not exist" message (DNE) along the reverse credit path. The message contains the addresses of the complete set of nodes that would store replicas of the content if it existed. Intermediate nodes can contact a member of this set to verify that the object does not exist. If they are convinced that the object really does not exist, they restore the confidence of the successor node to compensate for the reduction taken in step (1).

Each participating node has an incentive to follow each of the protocol steps: Node A wants to receive all the chunks, node Z wants to be credited for transmitting all the chunks, and all nodes wish to maintain the confidence of their predecessors along the credit path. When a node *defects* from the protocol at some stage, it can collect credit without providing the corresponding service. However, the price is a drop in the confidence of the node's predecessor. Also, the damage is limited to the size of a single chunk, which can be made appropriately small.

In general, for any failure, the client A is charged for at most a single chunk – a modest loss. The charge can be interpreted as the price for imposing load on the overlay by issuing a request that could not be satisfied. Such a charge also discourages flooding requests into the system; the client must pay for each and every request it makes. The client can minimize the loss associated with a failure when it begins with a small chunk and gradually increases the request size as its confidence in the path increases.

Over the long term, transitive trading tends to balance credit and debt among a node's overlay neighbors, maximizing the chances that the node will be able to obtain content in the future. Moreover, participation in a transitive trade is beneficial because it increases the confidence of each node along the path in its successor.

At the same time, nodes have a disincentive to refuse participation in a transitive trade. Such a refusal leads the predecessor along the credit path to reduce its confidence

in the node. While the failure of a neighbor adversely affects a node, if it happens repeatedly, the node quickly reduces its confidence in that neighbor, and avoids routing messages through that neighbor in the future. As a result, failing nodes are avoided by the neighbors and become isolated.

It is important that nodes are not penalized for being off-line. When a node is off-line, other nodes merely suspend their relationship with the node until it returns. A related question is whether a node has an incentive to swap credit from an established neighbor to a newcomer as part of a transitive trade. In practice, having credit with a large and diverse set of neighbors maximizes the chances that a node will be able to successfully locate a credit path for a future request.

3.4 Caching

In general, objects in a content distribution system have a highly skewed popularity distribution [20][3]. To avoid load imbalances as a result of such skew, caching is used in these systems to dynamically adjust the number of nodes serving a content object according to its popularity. Typically, once a node has obtained some content for itself, it serves the content to other interested clients from its local cache. Thus, popular objects tend to be replicated widely.

In Scrivener, dynamic caching is required to address an additional form of imbalance caused by skewed popularity. Without caching, nodes serving popular objects would tend to accumulate a huge amount of credit. Nodes that serve less popular objects would tend to accumulate debt and lack the "earning potential" to ever repay the debt. Our simulations (see Section 5) will demonstrate this effect in action and show how caching addresses the problem. Moreover, nodes have an incentive to cache objects, because it increases their earning potential. Caching popular objects allows a node to earn the credit needed to satisfy its own future needs.

4   Implementation

In this section, we describe an implementation of our Scrivener prototype. We chose to implement our prototype using FreePastry, a structured overlay network with a distributed hash table service called PAST [13, 30, 31]. Scrivener uses only the key-based routing (KBR) API [9] exported by FreePastry [13]. Thus, our implementation will also work with any structured overlay that supports this interface, e.g., Chord [34].

4.1   Background

Pastry is a structured p2p overlay network that provides a KBR service. In such overlays, every node and every object is assigned a unique identifier randomly chosen from a large id space, referred to as a *nodeId* and *key*, respectively. Given a message and a key, Pastry can route the message to the live node whose nodeId is numerically closest to the key in less than $\log_{2^b} N$ hops, where N is the number of nodes in the network and b is the routing base, usually set to 4. Castro et al. [6] describe techniques that make Pastry robust to collusions of a minority of malicious nodes in the overlay who attempt to compromise the overlay. These techniques are complementary to the techniques described

---

[3] This is not a problem for BitTorrent, since every user attempts to get the same object, and the popularity of each block is identical.

in this paper and can be used in conjunction with Scrivener if malicious participants (rather than mere freeloaders) are a threat.

PAST provides a distributed hash table (DHT) abstraction on top of Pastry. Each stored item in PAST is given a key (hereafter referred to as the *handle*), and replicas of an object are stored at the k live nodes whose nodeIds are the numerically closest to the object's handle (these nodes are called a *replica set*). PAST maintains the invariant that the object is replicated on k nodes, regardless of node addition or failure. If a node in the replica set is out of space, the object will be diverted to a node close in nodeId space but not in the replica set, and stored there temporarily. The handle is built from a cryptographically secure hash (e.g., SHA-1) applied to the data being stored. As such, the handle has sufficient information for the holder of the handle to verify that the content obtained from PAST is authentic.

4.2   Node bootstrapping

Recall that when a new node joins the system, it has no credit or debt. To earn credit, it needs to obtain some initial content that it can then serve to other nodes. In our prototype implementation, PAST's normal content placement and replication policy provides a node with its initial set of content objects.

When a PAST node joins the system, it is *required* to store a set of objects based on its position in the identifier space. The node obtains these initial objects from its neighbors in the id space for free; they form the new node's initial content offering and allow it to acquire credit with its overlay neighbors, which forward requests for these objects to the node as part of PAST's normal lookup operation. Our simulation results show that this simple mechanism suffices for a node to quickly bootstrap itself.

4.3   Finding credit paths

A key implementation issue is how to efficiently discover credit paths. The Pastry routing primitive finds an overlay path to a node that stores the requested content object, given the object's identifier. Finding a credit path introduces the additional constraint that each node along the path must be in good standing with its successor.

Our prototype uses a randomized, greedy algorithm to discover credit paths. To determine the next hop, a Scrivener node first selects the set of neighbors that satisfy the Pastry routing constraint. These nodes either have identifiers that match the requested object handle in a longer prefix than the present node's id, or their id matches as long a prefix as the present node's id but is numerically closer to the object handle. Forwarding the request to a node in this set guarantees that the route is loop-free and will end at a node that has the desired content, assuming the content exists in the overlay.

Next, we subtract from the candidate set any neighboring nodes where the present node is not in good standing. These neighbors would refuse requests from the present node because it had exceeded its credit limit. Because all of the information used by nodes to rate their neighbors is available equally to both parties, nodes can easily track their standing with their neighbors.

Among the set of remaining candidate nodes, we make a biased random choice, based on the following criteria:

– *Length of the neighbor's prefix match with the object handle.* Choosing a neighbor with higher prefix match than the present node reduces the latency and path length, and therefore also increases the chance to find a working path.

– *Confidence in the neighbor.* Neighbors with higher confidence values have been more helpful in the past, and are thus more likely to be helpful this time.
– *Amount of credit with the neighbor.* Choosing neighbors with higher credit helps the present node to balance credit and debt and therefore increases flexibility in handling future requests.

Scrivener strongly biases the forwarding choice toward neighbors with a prefix match (minimizing the number of overlay routing hops), while also trying to balance credit and debt, and gives preference to neighbors with high confidence values. More precisely, let $\mathscr{R}$ denote the remaining set of candidate nodes. Scrivener assigns a *score* to each node x in set $\mathscr{R}$, which is calculated as $\text{score}(x) = e^{\ell(x)} \cdot t(x) \cdot [c(x) - c_{\min} + 1]$, where $\ell(x) \geq 0$ is the number of additional digits that the neighbor x shares with the object handle relative to the present node, $c(x)$ and $t(x)$ are the credit and confidence value of neighbor x, and $c_{\min} = \min_{i \in \mathscr{R}} c(i)$. Then the probability that peer x is chosen is its score divided by the total score of all candidate peers, i.e., $\text{score}(x)/\sum_{i \in \mathscr{R}} \text{score}(i)$. The quality of a node's prefix match figures exponentially in its score to give a significantly greater weight to shorter routes. Note also that both confidence and credit/debt are measured in the same units, i.e., the number of objects or bytes transferred.

Our randomized, greedy algorithm is not guaranteed to discover a credit path even if one exists. A request could end up at a node that has no neighbor that satisfies the Pastry routing constraints and with which the node is in good standing. In such cases, the request cannot be forwarded on and the client will need to retry the request through a different neighbor.

Our simulations shows that the success rate is very high and the number of retries typically necessary to discover a credit path is very low in practice. There are several reasons for this. First, the Pastry overlay is richly connected and many redundant paths exist between a client and a node holding the required content. Second, dynamic caching effectively balances the "earning power" of nodes, avoiding strong imbalances in the credit available to different nodes. Third, the bias in the forwarding policy against nodes with low confidence tends to isolate freeloaders, causing requests to be effectively routed around such nodes. Lastly, the bias in the forwarding policy based on credit tends to balance the available credit a node has with its different neighbors. These various self-stabilizing forces reduce the probability that a credit path search might fail, either due to lack of credit or because a freeloader refuses to honor it.

4.4 Bounding lengths of credit paths

Unlike the native Pastry routing policy, Scrivener does not always choose a neighbor with a longer prefix match, even if such a neighbor exists. As a result, Pastry's logarithmic bound on the expected path lengths does not strictly hold. Note that shorter path lengths are desirable for two important reasons: (1) shorter path lengths ensure low delay and network utilization, and (2) shorter paths are more robust against node failures. Since the routing policy of Scrivener may occasionally lead to long paths, we resort to another mechanism to bound the path length.

In the prototype implementation, Scrivener artificially bounds the credit path length to be logarithmic in the overlay size. When the search for a credit path has reached this bound, the request is dropped. A rough estimate of the size of the overlay N suffices to determine the bound. Since nodeIds are assigned at random, the overlay size can be extrapolated from the local density of nodeIds with sufficient accuracy. When a search exceeds this boundary, the request is dropped. Our simulation results, presented in Section 5, show that the impact of this restriction on the ability to locate credit paths is minimal, while it ensures deterministic bounds on the system's resource consumption.
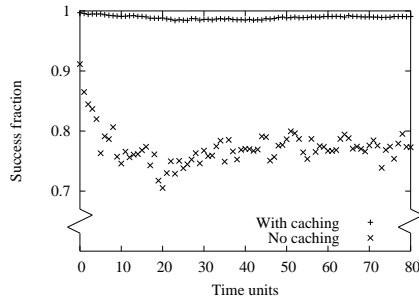
Fig. 3. Success rate with only obedient nodes.
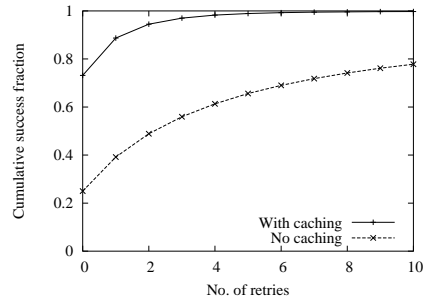

Fig. 4. Cumulative distribution of the number of retries to find a debt-based path.

5   Experimental results

In this section, we present simulation results to evaluate our prototype implementation. We simulate a system where network messages are delivered instantaneously. Objects are replicated using PAST's replication strategy, storing an object on the k nodes with nodeIds closest to the identifier for that object. When requesting an object, client nodes perform at most 10 queries, each time attempting to discover a credit path using the randomized greedy algorithm. The initial credit limit is set to 1 object, and increases linearly with the confidence the node has in its peer. The credit paths are limited to $\lceil 3 \log N \rceil$ hops. Each node also has a fixed sized, 1024-object soft cache to retain objects it has previously obtained to satisfy future requests. We implement an LRU cache replacement policy to replace entries from the cache when it is full.

A node's peers maintain their credit and confidence values for a node that is temporarily off-line. Also, the Pastry routing tables are persistent, i.e., a node remembers its table while it is off-line. Inappropriate entries are simply replaced by the existing overlay maintenance mechanisms, but biased towards peers with which the node already has a relationship. As a last resort, the node initiates a new relationship. Also, for each entry in the routing table, a node maintains at most three neighbors but uses only the one with the highest confidence value. (Confidence estimation is described in Section 3.2.)

5.1   Workload model

We use the model described by Gummadi et al. [20] to generate workloads. This model, derived from KaZaA traffic observations, captures the fetch-at-most-once behavior and the importance of new object arrivals in typical p2p file sharing applications. Based on this model, we chose the following parameters: number of nodes online $C = 800$, number of objects $O = 40,000$, request rate per node $\lambda_R = 50$, object arrival rate $\lambda_O = 12$, and node arrival rate $\lambda_C = 5$ (the units are nodes or objects per simulation time unit). The node departure rate is the same as the arrival rate, keeping the number of active nodes constant. Each object is initially replicated to $k = 3$ nodes. We assume that there is a fixed pool of 1,000 distinct nodes, out of which 800 are online at any time. As a result, during the first 40 time units all arriving nodes are fresh, but after time 40 all arriving nodes are those that were online once before. Nodes that go offline are chosen randomly from the currently live nodes.
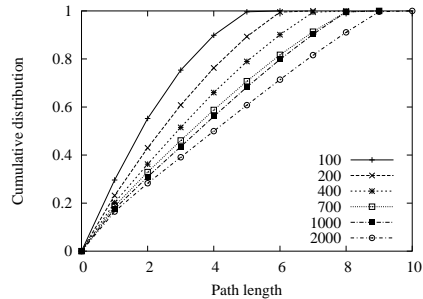
Fig. 5. Cumulative distribution of debt-based path lengths for different system sizes.

5.2 System performance

First, we study how our mechanisms affect the performance of the underlying cooperative content distribution system in the absence of freeloaders. In particular, we want to see how much overhead has been added to the system.

Success rate Figure 3 shows the fraction of successful requests, both with and without caching. Without caching, the success rate stabilizes around 80%. This is because object popularity is so uneven that nodes around the replicas of popular objects become indebted to the replica holders, making it sometimes impossible for a node to find a credit path to the replicas. Many requests to popular objects fail despite retries. However, allowing nodes to serve cached objects eliminates this problem and the success rate approaches 100%. The stability of the success rate suggests that the system balances out nicely and obedient nodes do not build up debt over time[4].

Figure 4 shows the number of retries required to successfully find a credit path. When caching is enabled, over 73% of queries succeed on the first attempt, and three attempts are sufficient to achieve over 95% success rate. We conclude that the policy enforcement in Scrivener with bounded paths does not seriously affect object fetch reliability in the absence of freeloaders.

Path efficiency Scrivener's randomized greedy routing strategy attempts to use Pastry's routing mechanism to achieve logarithmic-length paths, when possible, and falls back to less efficient mechanisms, when necessary, that are artificially capped to preserve an $O(\log N)$ expected path length (see Section 4.4). A cumulative distribution of path lengths at different overlay sizes is shown in Figure 5. By observing horizontal slices through this graph, we see that the growth in path length follows roughly the log of the number of nodes. Our simulations show that common case routes are quite efficient and the worst case routes are only twice as long as common-case routes.

Due to limitations of our simulation environment, we were unable to run simulations for overlay sizes larger than 2000. In order to emulate the effect of larger overlay sizes, we ran simulations with 1000 nodes, but with Pastry's routing base set to $b = 2$ instead of 4. The results show that the median Scrivener path lengths is around 5, close to the expected Pastry path length ($\log_{2^2} 1000 \approx 4.98$). Note that 5 is the expected path length for a Pastry overlay with one million nodes when $b = 4$. This result suggests that

---

[4] We have also implemented *speculative caching*, where nodes observe the requests they have forwarded and actively fetch objects that they consider popular. However, the improvements we observed in terms of success rate were insignificant.
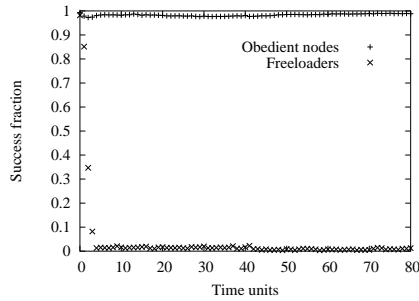
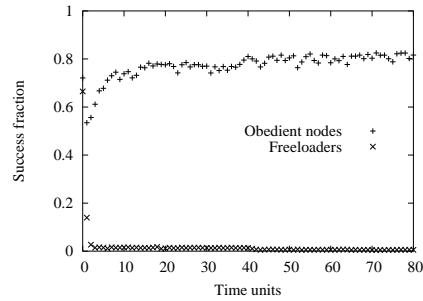**Fig. 6.** Success rate with 5% freeloaders that do not serve objects.

Fig. 7. Success rate with 50% freeloaders that do not serve objects.

Scrivener's greedy routing strategy easily scales to much larger overlay sizes than we were able to simulate.

Still, these longer paths, which would also occur as the number of nodes in the overlay increases, raise concerns about path usability, particularly if the system is experiencing high node churn. More nodes in a path increase the odds that one of those nodes will fail while a transitive trade is in progress. However, the system provides incentives for nodes to stay online until a transitive trade in which they are involved completes (see Section 3.3). If a path fails, the original requesting node can restart the trading protocol, find a new path to the source of the data (or a replica), and resume downloading the missing data.

The total overhead for Scrivener to fetch an object is the product of the average number of attempts to discover a credit path (≈ 2) and the average credit path length ($< \lceil 3 \log N \rceil$). Among competing systems that use auditor sets, KARMA [36] is the most efficient system we are aware of. KARMA's asymptotic message overhead is comparable to Scrivener's, but requires expensive public-key cryptographic operations and additional means of incentivizing auditors [36].

5.3 Introducing freeloaders

Next, we introduce freeloaders into our simulation. Freeloaders issue requests like obedient nodes, but they may refuse to serve objects. In a deployed system, freeloaders can be expected to attempt a variety of strategies. In the following experiments, we consider a number of freeloading strategies, and show that in all cases there are no sustainable benefits to freeloading. We simulate 800 nodes, but now with 5% freeloaders. We assume that freeloaders forward requests and participate in transitive trades, as this allows them to earn confidence with minimal traffic overhead. While obedient nodes undergo churn as specified in the model, freeloaders are always online throughout the entire simulation period. Recall that routing tables are persistent, ensuring that freeloaders cannot neither escape a bad reputation by periodically departing from the system nor by repeatedly exploiting the limited credit granted by obedient nodes looking to establish relationships.

Freeloaders that never serve First we consider freeloaders that never serve any object. Figure 6 shows that their success rate drops to below 5% within a few time units, yet that of obedient nodes is unaffected. Note that the success rate for freeloaders never
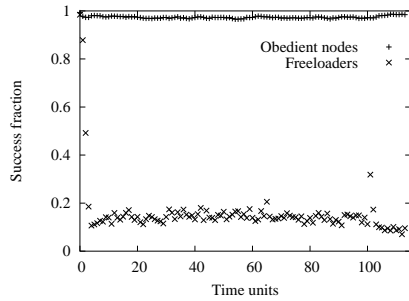
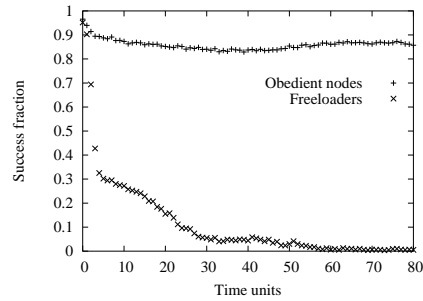**Fig. 8.** Success rate with a higher churn rate.

Fig. 9. Success rate with the worst-case scenario where every obedient node gives a high initial confidence to all freeloaders.

goes to zero. This is because freeloaders can still get the objects that they themselves are storing "for free."

To determine Scrivener's sensitivity to the size of the soft cache, we vary the cache size. The success rate remains virtually constant down to a cache size of 320 objects, and gradually decreases to 91% at 128 objects. This shows that Scrivener does not require a large soft cache to work efficiently.

We increased the fraction of freeloaders to 50%, with results shown in Figure 7. The success rate of freeloaders again drops quickly to near zero, while that for obedient nodes starts below 60% and plateaus at 80%. Note that with 50% freeloaders and a replication factor $k = 3$, it is expected that 12.5% of the objects are only stored by freeloaders and will thus never be served. This suggests that a more expensive search may increase the success rate somewhat, but with diminishing returns.

To test the system under extreme conditions, we increase the fraction of freeloaders to 80%. At this point, more than half of the objects are stored only by freeloaders and, unsurprisingly, the success rate for obedient nodes is only 30%. Also, as a result of more transitive trading failures, it takes longer for the success rate of obedient nodes to stabilize. Scrivener does continue to function remarkably well, despite the extreme freeloading rate. Given that these freeloaders receive no benefit from being present in the network, one would expect them to depart, allowing the remaining obedient nodes to operate more efficiently.

Since it takes time for obedient nodes to recognize freeloaders, one concern is that a high churn rate might enable freeloaders to get a satisfactory success rate by exploiting new node arrivals. We simulated a system with 800 nodes, but a churn rate λ_C of 50 nodes per time unit and with fresh nodes arriving for the first 100 time units. After time 100, the arriving nodes have all previously been part of the network and gone offline. Figure 8 clearly shows that with this higher churn of fresh nodes, the success rate for freeloaders stabilizes at around 15%, dropping after time 100 when the returning nodes remember previous freeloaders. Thus, while freeloaders can exploit newcomers, the benefit is limited. More importantly, the success rate for obedient nodes is unaffected. While obedient nodes waste some effort handling requests from freeloaders, they give clear priority to serving each other.

Recall that a Scrivener node grants an initial credit to its chosen neighbors. We next consider an attack where a freeloader somehow convinces an obedient node to choose it as a neighbor, thus granting it an initial credit. We consider a worst-case
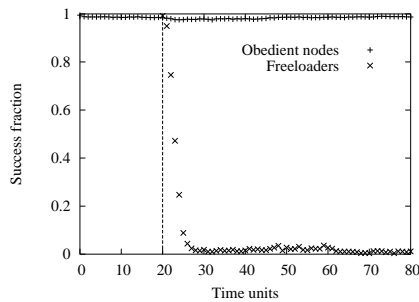
Fig. 10. Success rate with freeloaders that participates in transitive trades but do not fetch objects for the first 20 time units.
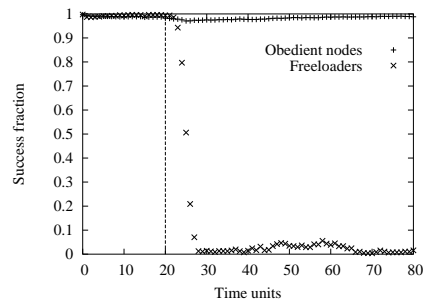
Fig. 11. Success rate with freeloaders that serve objects only for the first 20 time units.

scenario where freeloaders can always manipulate obedient nodes into choosing them as neighbors. With such an attack, freeloaders could now exploit the initial credit from each obedient node. Figure 9 shows that, indeed, freeloaders get a better success rate initially. However, the success rate drops to 30% quickly and gradually goes down as obedient nodes refuse to serve freeloaders after their debts build up. Our simulations show that, even with such a hypothetical attack, freeloaders would have little benefit and obedient nodes would observe no significant change in their own success rate.

Short-term cooperation Participation in transitive trades, alone, can earn confidence and increase credit limits without actually serving any object. An interesting question is whether it is possible for freeloaders to build up confidence simply by participating in transitive trades, and then exploit that confidence. In Figure 10, we simulate freeloaders that participate in transitive trades for 20 time units before fetching any object. The success rate for freeloaders drops to below 0.1 within ten time units. Thus, participation in transitive trades does have a benefit, but only a small one.

We also simulated nodes that were obedient for 20 time units and then began freeloading. As shown in Figure 11, the freeloader's success rate now takes seven time units to drop below 0.1. The freeloader does benefit from its earlier obedience. However, once freeloading behavior begins, the success rate remains high for only two time units, then falls quickly.

These experiments demonstrate that short-term cooperation is not an effective strategy for freeloaders to exploit the system; once they start to freeload, obedient nodes will quickly refuse to serve them.

Providing partial service Another possible freeloading behavior is to serve objects at a reduced rate. We first consider freeloaders that arbitrarily serve half of their requests. Figure 12 shows that the success rate for freeloaders drops to and remains at roughly 50% — the same rate at which they are providing service. Note also that the number of objects received by freeloaders also approaches and stabilizes at the same level as the number they serve.

Another potential strategy is to have a target quality of service. This freeloading behavior serves only enough requests to maintain a desired success ratio. We simulate freeloaders that target a 50% success rate. Figure 13 shows that the resulting success rate oscillates around 50%. As before, the number of objects served by the freeloader quickly dictates the number of objects the freeloader is allowed to consume.
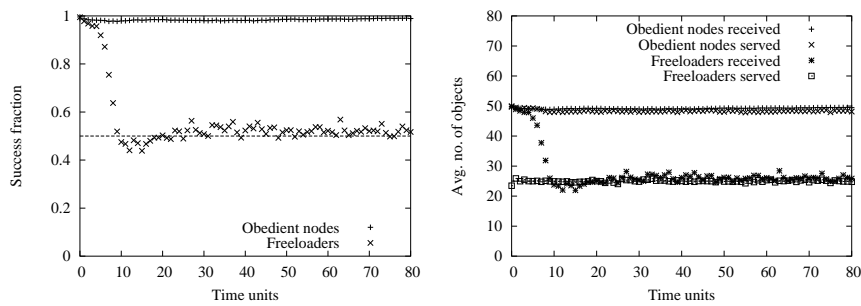
Fig. 12. Success rate and number of objects served and fetched with freeloaders that serve half of the object requests.
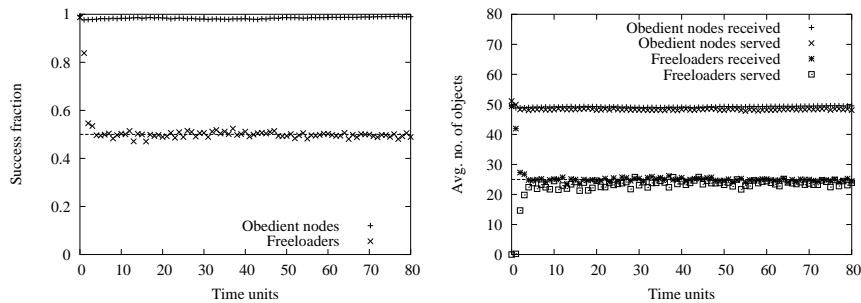

Fig. 13. Success rate and number of objects served and fetched with freeloaders that aim at 50% success rate.

We finally consider a strategy that alternates between obedience and freeloading, changing behaviors every 20 time units. Figure 14 shows that the success ratio quickly tends toward 1 and 0 whenever these nodes switch to cooperation and to freeloading, respectively, with the peak success ratio dropping over time. Also, during the cooperation periods, the former freeloaders service more requests, effectively making up for the debts they previously accumulated. On average, this alternation strategy performs worse, from the freeloader's perspective, than the previous 50% service strategy.

Other experiments In our simulation, a node requests 50 objects per time unit. If each object is 64 Kbytes, this translates into roughly 3MB of data per time unit — about the size of a typical MP3 file or digital photograph. If we consider users that attempt to download 100MB of data per day, their success rate would drop to zero in about an hour. Increasing the download rate does not help, since its merely accelerates the decline in success rate.

To test Scrivener's sensitivity to the size of the downloaded content, we ran simulations where we divided large objects into smaller chunks that were stored and downloaded separately. The success rate of obedient nodes improved relative to our earlier experiments. When downloading smaller chunks, smaller credits were necessary, increasing the success rate of transitive trading. Also of note, freeloaders experienced an even lower success rate. Because a desired object may now be spread over several chunks, the odds successfully obtaining all of a file's chunks diminished. Of course,
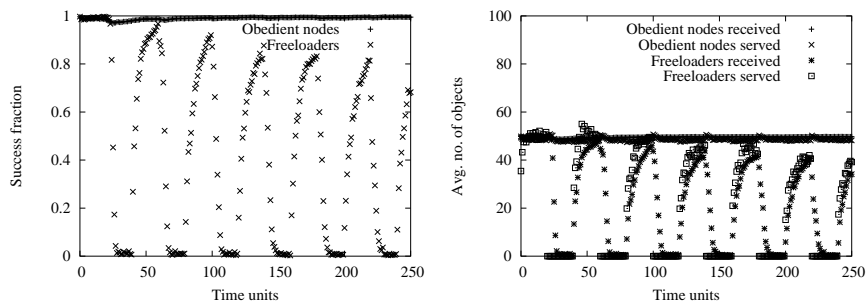
Fig. 14. Success rate and number of objects served and fetched with freeloaders that switch between cooperation and freeloading every 20 time units.

breaking a file into chunks will increase the overhead rate, as each chunk will need to be separately located and fetched.

We have also simulated scenarios with obedient nodes with diverse bandwidth capacities. The success rate for both types of nodes are very close to 100%, although the success rate for high-end nodes drops slightly. This shows that Scrivener can accommodate modest imbalances in the demands and "earning potentials" of participating nodes gracefully. Other approaches, including treating a high-end node as several virtual nodes, may also be applicable.

Discussion   We have evaluated mechanisms to make bandwidth-limited p2p content distribution networks robust against freeloaders. Obedient nodes experience modest additional overhead, and over a variety of freeloading behaviors, freeloaders achieve only the level of service that they willing to provide to others in the network, even for large numbers of freeloaders in the system. Our simulations demonstrate that the obedient strategy maximizes a node's utility, i.e., Scrivener appears to be economically strategy-proof.

While our simulation environment does not model delay, the modest increase in the path length of content requests, combined with the fact that most p2p content downloads are bandwidth-limited, strongly suggests that download delay is not significantly affected by Scrivener.

We note that freeloaders still get some benefit during the first few time units after they join the system. If a freeloader can create new identities without restriction, such "Sybil attacks" [11] would be able to defeat our mechanisms. As discussed in Section 2, we require that the p2p overlay has security features to prevent such attacks. Alternatively, Scrivener could adopt a policy where all nodes receive degraded service quality when they join the p2p network, with the quality improving only after the new node has proven its worth.

6   Related work

There has been much work on providing incentives for cooperation in distributed systems. We roughly categorize the related works as follows.

Bandwidth-sharing networks   SLIC [35] considers the query nature of unstructured p2p systems like Gnutella [17]. It proposes giving nodes service levels proportional to

their contribution, so as to provide nodes incentives to share more data and handle more traffic. BitTorrent [7] facilitates large numbers of nodes all trying to acquire exactly the same file, with an emphasis on very large files (e.g., software distributions, digital movies, and so forth). Every BitTorrent node will have acquired some subset of the file and will trade blocks with other nodes until it has the whole file. In order to bootstrap new nodes, nodes reserve 1/4 of their bandwidth for altruistic service. Nodes that fairly trade their bandwidth will experience a higher quality of service. Anagnostakis and Greenwald [3] suggested that performance can be improved if exchanges are extended to allow involving multiple parties. Scrivener solves the more general problem, where nodes are interested in content from a large set, of potentially much smaller size. We allow nodes to acquire credits from the files they serve to obtain any other files they desire in the future. Thus, they have an incentive to serve, even when they themselves do not require any content at the moment.

GNUnet [19] uses the idea of locally-maintained debit/credit relations in a similar fashion to our own work. It also uses debt relationships across nodes, comparable to our debt-based routing. As GNUnet is more concerned with anonymity than network efficiency, it does not support transmitting objects directly across the network. All traffic goes through the overlay, forcing intermediate nodes to carry the bulk traffic of the object transfer while giving them no particular incentive to do this, save for maintaining their own anonymity. For a path with n nodes, GNUnet transfers the object $O(n)$ times. Scrivener, on the other hand, finds efficient routes and transmits bulk data directly over the Internet, yielding higher performance, but lacking GNUnet's anonymity features. Scrivener also provides mechanism to locate and fetch objects, leveraging its existing credit/debit framework.

Storage networks In a storage network, nodes share spare disk capacity for applications such as distributed backup systems. Ngan et al. [27] propose an auditing mechanism, which allows cheaters to be discovered and evicted from the system. Samsara [8] enforces fairness by requiring an equal exchange of storage space between peers and by challenging peers periodically to prove that they are actually storing the data. Storage incentivicing systems are solving a fundamentally different problem than bandwidth incentivicing systems. Storage is a commitment, over a long time period, to provide a stable service. If misbehavior is detected, a node can punish another by simply deleting its files. Bandwidth, on the other hand, is an ephemeral service. Bits transmitted cannot be taken back. Retribution can only be taken by refusing future requests.

Reputation Resource allocation and accountability problems are fundamental to p2p systems. Dingledine et al. [10] surveys many schemes for tracking nodes' reputations. In particular, if obtaining a new identity is cheap and positive reputations have value, negative reputation could be shed easily by leaving the system and rejoining with a new identity. Friedman and Resnick [14] also study the case of cheap pseudonyms, and argue that suspicion of strangers is costly. There have been attempts to build a distributed trust management system [1,22]. Blanc et al. suggest a reputation system for incentivicing routing in peer to peer networks that uses a trusted authority to manage the reputation values for all peers [4]. Unlike those efforts, our design relies solely on locally observable (and thus more trustworthy) information.

Trading and payments SHARP [16] is a framework for distributed resource management, where users can trade resources like bandwidth with trusted peers. KARMA [36] and SeAl [29] rely on auditor sets to keep track of the resource usage of each participant in the network, similar to Ngan et al.'s quota manager approach [27]. MojoNation [26] similarly allowed peers to exchange certificates for resources. Golle et al. [18]

considered centralized p2p systems with micro-payments, analyzing how various user strategies reach equilibrium within a game theoretic model.

Trading and payments architectures may be too expensive for many content distribution systems, as each download would incur cryptographic operations and additional communication. Moreover, implementing micro-payments either requires a centralized authority to issue currencies, or uses distributed trust and currency, which is still an active research area.

Mobile ad hoc networks Since nodes in mobile ad hoc networks rely on each other to forward traffic, incentives are as important in these networks as they are in p2p content distribution systems. Marti et al. [25] consider monitoring the performance of other nodes and routing around uncooperative nodes. CONFIDANT [5] is a distributed reputation system to detect and isolate misbehaving nodes. Salem et al. [32] propose a micro-payment architecture for multi-hop cellular networks. Catch [24] is a mechanism to identify and punish selfish nodes who do not forward packets in a multi-hop wireless setting based on an anonymous challenge-response protocol. In general, mobile ad hoc networks may require different incentive mechanisms than p2p systems due to their limited computational resources and peer connectivity.

7 Conclusions

This paper presents Scrivener, a decentralized system that provides nodes in a cooperative content distribution network with incentives to share their bandwidth resources. Scrivener only requires nodes to track their neighbor's behavior. It uses a greedy randomized routing algorithm to find a credit path, allowing a node to leverage credit is has with its overlay neighbors to obtain content from an unrelated node that holds the desired content. At the same time, Scrivener effectively prevents freeloaders from exploiting obedient nodes. Our results show that Scrivener is scalable and effective at deterring freeloading behavior while incurring modest overhead.

References

1. K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *Proc. of the 10th Int'l Conf. of Information and Knowledge Management*, Atlanta, GA, 2001.
2. E. Adar and B. Huberman. Free riding on Gnutella. *First Monday*, 5(10), Oct. 2000.
3. K. G. Anagnostakis and M. B. Greenwald. Exchange-based incentive mechanisms for peer-to-peer file sharing. In *Proc. 24nd Int'l Conf. on Distributed Computing Systems*, Washington, DC, Mar. 2004.
4. A. Blanc, Y.-K. Liu, and A. Vahdat. Designing Incentives for Peer-to-Peer Routing. In *Proc. 24th IEEE Infocom*, Miami, FL, Mar. 2005.
5. S. Buchegger and J.-Y. Le Boudec. Performance analysis of the CONFIDANT protocol. In *Proc. MobiHoc'02*, Lausanne, Switzerland, June 2002.
6. M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Security for structured peer-to-peer overlay networks. In *Proc. OSDI'02*, Boston, MA, Dec. 2002.
7. B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Econ. of Peer-to-Peer Systems*, Berkeley, CA, June 2003.
8. L. P. Cox and B. D. Noble. Samsara: Honor among thieves in peer-to-peer storage. In *Proc. SOSP'03*, Bolton Landing, NY, Oct. 2003.
9. F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common API for structured peer-to-peer overlays. In *Proc. IPTPS'03*, Berkeley, CA, Feb. 2003.
10. R. Dingledine, M. J. Freedman, and D. Molnar. Accountability. In A. Oram, editor, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, 2001.

11. J. R. Douceur. The Sybil attack. In *Proc. IPTPS'02*, Cambridge, MA, Mar. 2002.
12. J. Feigenbaum and S. Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Proc. 6th Int'l Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, Atlanta, GA, Sept. 2002.
13. FreePastry. Open source implementation of Pastry. `http://freepastry.rice.edu/`.
14. E. Friedman and P. Resnick. The social cost of cheap pseudonym. *Journal of Economics and Management Strategy*, 10(2):173–199, 2001.
15. K. Fu, M. F. Kaashoek, and D. Mazières. Fast and secure distributed read-only file system. *ACM Transactions on Computer Systems*, 20(1), Feb. 2002.
16. Y. Fu, J. S. Chase, B. N. Chun, S. Schwab, and A. Vahdat. SHARP: An architecture for secure resource peering. In *Proc. SOSP'03*, Bolton Landing, NY, Oct. 2003.
17. Gnutella. `http://www.gnutella.com/`.
18. P. Golle, K. Leyton-Brown, I. Mironov, and M. Lillibridge. Incentives for sharing in peer-to-peer networks. In *Proc. 3rd ACM Conf. on Electronic Commerce*, Tampa, FL, Oct. 2001.
19. C. Grothoff. An excess-based economic model for resource allocation in peer-to-peer networks. *Wirtschaftsinformatik*, June 2003.
20. K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. SOSP'03*, Bolton Landing, NY, Oct. 2003.
21. G. Hardin. The tragedy of the commons. *Science*, 162:1243–1248, 1968.
22. S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The EigenTrust algorithm for reputation management in p2p networks. In *Proc. WWW 2003*, Budapest, Hungary, May 2003.
23. KaZaA. `http://www.kazaa.com/`.
24. R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Sustaining Cooperation in Multi-hop Wireless Networks. In *Proc. NSDI'05*, May 2005.
25. S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proc. MobiCom'00*, Boston, MA, Aug. 2000.
26. MojoNation. `http://en.wikipedia.org/wiki/MojoNation/`, see also Mnet `http://mnetproject.org`.
27. T.-W. J. Ngan, D. S. Wallach, and P. Druschel. Enforcing fair sharing of peer-to-peer resources. In *Proc. IPTPS'03*, Berkeley, CA, Feb. 2003.
28. T.-W. J. Ngan, D. S. Wallach, and P. Druschel. Incentives-compatible peer-to-peer multicast. In *2nd Workshop on the Economics of Peer-to-Peer Systems*, Cambridge, MA, June 2004.
29. N. Ntarmos and P. Triantafillou. SeAl: Managing accesses and data in peer-to-peer sharing networks. In *Proc. of the 4th IEEE Int'l Conf. on P2P Computing*, Zurich, Switzerland, 2004.
30. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object address and routing for large-scale peer-to-peer systems. In *Proc. Middleware*, Heidelberg, Germany, Nov. 2001.
31. A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proc. SOSP'01*, Oct. 2001.
32. N. B. Salem, L. Buttyan, J.-P. Hubaux, and M. Jakobsson. Node cooperation in hybrid ad hoc networks. *IEEE Transactions on Mobile Computing*, 2005. To appear.
33. J. Shneidman and D. Parkes. Rationality and self-interest in peer to peer networks. In *Proc. IPTPS'03*, Berkeley, CA, Feb. 2003.
34. 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. SIGCOMM'01*, San Diego, CA, Aug. 2001.
35. Q. Sun and H. Garcia-Molina. SLIC: A selfish link-based incentive mechanism for unstructured peer-to-peer networks. In *Proc. 24nd Int'l Conf. on Distributed Computing Systems*, Washington, DC, Mar. 2004.
36. V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer. KARMA: A secure economic framework for p2p resource sharing. In *Workshop on Econ. of Peer-to-Peer Systems*, Berkeley, CA, June 2003.
37. M. Waldman and D. Mazières. Tangler: A censorship-resistant publishing system based on document entanglements. In *Proc. ACM CCS*, Philadelphia, PA, Nov. 2001.