

Robust and Efficient Incentives for Cooperative Content Distribution

Michael Sirivianos Xiaowei Yang Stanislaw Jarecki
 Duke University Duke University University of California, Irvine
 msirivia@cs.duke.edu xwy@cs.duke.edu stasio@ics.uci.edu

Abstract—Content distribution via the Internet is becoming increasingly popular. To be cost-effective, commercial content providers are now using peer-to-peer (P2P) protocols such as BitTorrent to save bandwidth costs and to handle peak demands. When an online content provider uses a P2P protocol, it faces an incentive issue: how to motivate its clients to upload to their peers.

This paper presents Dandelion, a system designed to address this issue. Unlike previous incentive-compatible systems, such as BitTorrent, our system provides non-manipulable incentives for clients to upload to their peers. A client that honestly uploads to its peers is rewarded in the following two ways. First, if its peers are unable to reciprocate its uploads, the content provider rewards the client’s service with credit. This credit can be redeemed for discounts on paid content or other monetary rewards. Second, if the client’s peers possess content of interest and have appropriate uplink capacity, the client is rewarded with reciprocal uploads from its peers.

In designing Dandelion, we trade scalability for the ability to provide robust incentives for cooperation. The evaluation of our prototype system on PlanetLab demonstrates the viability of our approach. A Dandelion server that runs on commodity hardware with a moderate access link is capable of supporting up to a few thousand clients. The download completion time for these clients is substantially reduced due to the additional upload capacity offered by strongly incentivized uploaders.

Index Terms—Peer-to-peer, content distribution, incentives, fair-exchange, symmetric cryptography.

I. Introduction

Content distribution via the Internet is becoming increasingly popular among the entertainment industry and the consumers alike. An attractive approach for commercial online content distribution is the use of peer-to-peer (P2P) protocols. This approach does not require a content provider to over-provision its bandwidth to handle peak demands, nor does it require the provider to rely solely on purchased service from a third-party such as Akamai. Instead, a P2P protocol such as BitTorrent [1] harnesses its clients’ unused uplink bandwidth, and saves the bandwidth and computing resources of a content provider. Huang et al. [2] showed that peer-assisted content distribution can substantially reduce the operating costs of Video on Demand services. To that effect, BBC has successfully launched its iPlayer peer-assisted VoD service and leading content providers have now partnered with BitTorrent, Inc [3]. This trend indicates that P2P protocols enable a site to cost-effectively distribute content.

When an online content provider uses a P2P protocol, it faces an incentive issue: how to motivate clients that possess content to upload to others. This issue is of paramount

importance because the performance of a P2P network is highly dependent on the users’ willingness to contribute their uplink bandwidth. In addition, in a competitive market, a content provider with paying customers needs to offer better quality of service guarantees than the ones offered by free P2P content distribution systems. However, selfish (rational) users tend not to share their bandwidth without external incentives [4]. Although the popular BitTorrent protocol, has incorporated the rate-based tit-for-tat incentive mechanism, this mechanism bears two weaknesses. First and foremost, it does not encourage clients to seed, i.e., to upload to other peers after completing the file download. Second, it is vulnerable to manipulation [5–8], allowing modified clients to free-ride and still achieve download rates equal to or higher than the ones of cooperative clients (§ II-B,VI-C).

In previous work [9], we introduced Dandelion, a protocol that provides provably non-manipulable incentives for seeding and is not susceptible to free-riding. Although the protocol was shown to be sufficiently scalable, its incentive mechanism was completely centralized. In this paper, we built upon our initial design and propose changes that *partially* decentralize the protocol. These changes render Dandelion more scalable, while they maintain its original desirable properties. Our modified protocol provides robust incentives using two mechanisms.

The first mechanism, *credit-based-exchange*, guarantees strict fair-exchange of content uploads for real monetary value. This mechanism is useful when a client has content that interests its peer but the peer has no content of interest to reciprocate with. *Selfish* clients (i.e., rational clients that do not upload unless they expect to be rewarded) earn credit when they upload valid content to their peers. Credit can be redeemed at a content provider for discounts on the content or for other types of monetary awards. Given appropriate pricing schemes, we expect that a selfish client is motivated to serve content to its peers. The second mechanism, *Tit-for-tat-based (TFT-based) exchange*, renders the protocol more scalable by partially decentralizing it. It enables clients that are mutually interested in each other’s content to barter their uplink bandwidth.

A key challenge lies in making the exchange of content uploads for credit efficient and practical, while robust to manipulation. Practice has shown that this problem is a major stumbling block for the commercial adoption of micropayment-based incentive schemes [10]. Manipulability may make content distributors wary of substantial monetary losses in case the client software is compromised.

We address this challenge based on the insight that the

content provider itself is a trusted third party (TTP) and can mediate the content exchange between its clients. Under the credit-based exchange protocol, clients exchange data for credit and a server mediates the transaction. The server uses only efficient symmetric cryptography on critical data paths and sends only short messages to its clients.

In our setting, unfairness during TFT exchanges results in monetary losses for the peer that does not receive its deserved reciprocation. We address this issue using an optimistic fair-exchange protocol that is an adaptation of BAR Gossip [11] and classic optimistic fair-exchange exchange schemes [12, 13]. In optimistic fair-exchange, the trusted third party is involved only when an error occurs or when dishonest participants do not follow the protocol.

Our work makes the following contributions:

- 1) The design of Dandelion, a novel hybrid incentive scheme for commercial P2P content distribution, which is based on efficient cryptographic fair-exchange protocols.
- 2) The prototype implementation of a Dandelion-based system that is suitable for P2P distribution of static content.
- 3) The evaluation of our implementation on PlanetLab [14], which identifies the scalability limits of our incentive mechanism and demonstrates the plausibility of our approach.

The rest of this paper is organized as follows. § II provides background and motivates our design. § III provides an overview of Dandelion and describes the system model under which it is designed to operate. § IV describes the design of Dandelion and discusses its properties. § V describes the implementation of our prototype system. § VI presents the experimental evaluation of our implementation. In § VII we discuss prior work and we conclude in § VIII.

II. Background

Dandelion’s design addresses the incentive issues in P2P content distribution protocols such as BitTorrent [1] and eMule [15]. In this section, we motivate the design of Dandelion by discussing the weaknesses of BitTorrent’s incentive mechanism.

In the rest of this paper we use a BitTorrent-like terminology. A *seeder* refers to a client that uploads to its peers after it has completed its download. A *leecher* is a client that has not completed its download. A *free-rider* refers to a client that downloads content from other peers without incurring any cost, i.e. without uploading content or without expending currency. A *swarm* refers to all clients that actively participate in the protocol for a given content item. The *choking algorithm* refers to the client-side function of selecting peers to upload content to (unchoke) in parallel, based on a predetermined criterion. *Optimistic unchoking* refers to temporarily unchoking a peer, although that peer does not currently satisfy the unchoking criterion.

A. Impact of Seeding

The popular BitTorrent protocol employs the rate-based “tit-for-tat” (TFT) incentive mechanism [1]. A peer prefers to upload to (unchoke) another peer that reciprocally uploads parts of the same file. This mechanism mitigates free-riding,

but does not provide explicit incentives for seeding. Although several BitTorrent deployments rely on clients to honestly report their uploading history [16], and use this history to decide which clients can join a swarm, practice has shown that clients can fake their upload history [10, 17, 18] or collude [19].

Seeders improve download completion times, because they increase the content availability and the aggregate upload bandwidth. In addition, incentives for seeding are crucial because a large portion of P2P clients reside behind asymmetric Internet links. This means that the total upload capacity of the P2P network may be substantially lower than its total download capacity. However, lack of incentives leads to BitTorrent swarms being underprovisioned in terms of seeders [20]. In order to rectify this situation one needs to persuade peers to remain online to seed after they complete their download. In § VI-B2, we show that the download rates of leechers in BitTorrent swarms increases substantially as the number of clients that seed increases. These observations are corroborated by previous measurements in P2P content distribution systems [20, 21].

B. Free-riding in BitTorrent

A general observation is that since BitTorrent’s tit-for-tat incentives reward cooperative leechers with improved download times, leechers are always incentivized to upload. This observation relies on the assumption that users aim only at maximizing their download rates. However in practice, BitTorrent users may be reluctant to upload even if uploading improves their download times. For example, users with access providers that impose quotas on outgoing traffic or users with limited uplink bandwidth (e.g., 1.5Mbps/128Kbps ADSL) may wish to save their uplink for other more critical tasks.

Considering the trade-off between performance and susceptibility to free-riding [22], BitTorrent purposely does not implement a strict TFT strategy. In particular, it employs rate-based instead of chunk-level TFT, and BitTorrent clients optimistically unchoke peers for relatively long periods of time (30 seconds). Furthermore, BitTorrent seeders select peers to upload to regardless of whether those peers upload to others.

Based on the above observations and previous work on BitTorrent exploitation [5, 6], we employ the “large view” exploit [7] to free-ride in BitTorrent-like swarms. The free-rider client obtains a larger than normal view of the network and connects to all peers in its view, while it does not upload any content. Using this exploit in a sufficiently large swarm, a free-rider can find more seeders, which do not employ tit-for-tat. It can also increase the frequency with which it becomes optimistically unchoked, compared to a compliant client, which typically connects to 50-100 peers. In § VI-C, we extend our free-riding study to further motivate our design. We experiment with free-riders in larger PlanetLab-residing torrents comprising of ~ 400 leechers and under more realistic bandwidth distribution. We also investigate how the existence of seeders affects the effectiveness of the exploit.

Our results suggest that the large view exploit has the potential to be widely adopted because it is beneficial for free-riders. A dire prediction is that if more and more users that

are reluctant to upload employ free-riding clients, BitTorrent communities will experience the “tragedy of the commons,” until those users realize that they need to use cooperative clients in order to improve their download rates. Dandelion’s non-manipulable incentives explicitly address this issue by preventing free-riders from obtaining any content without reciprocating or spending money.

III. Overview and System Model

We now provide an overview of Dandelion and describe the system model under which it is designed to operate. In addition, we introduce our setting and notation.

A. System Overview

The Dandelion *server* acts as a tracker redirecting its *clients* to other clients that are able to serve their requests for content. The content provider splits content into verifiable *chunks*, and clients exchange carefully selected chunks. The content provider deploys in addition to the server, at least one client with the complete content (initial seeder).

Dandelion employs a hybrid incentive mechanism. In case a client has content that interests a peer, but that peer does not have content that interests the client, the system entices the selfish client to upload by rewarding the client with credit. The system also rewards a selfish client with credit when the peer is unable to reciprocate at the rate it downloads from the client. The server maintains the credit balance of each of its clients and converts credit to monetary rewards, such as discounts on paid content. To ensure that no user can be dishonest in the content-for-credit transactions, we employ a fair-exchange mechanism based on symmetric key cryptography. This mechanism requires the involvement of a trusted third party in each transaction. We refer to this mechanism for exchange of content uploads for credit as *credit-based exchange* and the chunks that are uploaded under this mechanism as *credit-traded*.

A Dandelion client employs a tit-for-tat mechanism when its peers can reciprocate with content of interest. That is, the client uploads content to a peer at the same rate that the peer uploads content to the client. However, a simple tit-for-tat scheme, such as BitTorrent’s, is susceptible to the “large view” exploit. Free-riders that connect to many peers in their swarm can benefit considerably by their peers’ initial offers. To address this issue we employ an optimistic fair-exchange mechanism [12, 13] based on public key cryptography. Optimistic fair-exchange requires the involvement of a trusted third party only in case a peer misbehaves. We refer to the tit-for-tat mechanism as *TFT-based exchange* and the chunks that are uploaded under this mechanism as *TFT-traded*.

B. System Model

We assume two types of clients, which we define as follows:

- *Selfish* (rational) clients strategize based on a utility function that describes the cost they incur when they upload a chunk to their peers and when they pay credit to download a chunk. It also describes the benefit they gain when they are rewarded with credit for correct chunks they upload. A selfish client aims at maximizing its utility.

A selfish client may consider manipulating the credit system in order to increase its utility by misbehaving as follows: a) upload no chunks to a peer, and yet claim credit; b) upload garbage either on purpose or due to communication failure to a peer, and yet claim credit or be reciprocated with valid content by the peer; c) download chunks from selfish clients, and yet attempt to avoid being charged or reciprocating with chunks; d) attempt to download chunks from selfish peers that are not interested in its content without having sufficient credit; and e) attempt to boost its credit by colluding with other clients or by opening multiple Dandelion accounts.

- *Malicious* clients may be faulty or strategize based on irregular utility functions, e.g., their utility may increase by harming others, despite not obtaining credit or content. They may misbehave as follows: a) attempt to make honest clients appear as malicious or dishonest, or attempt to cause them to be charged for chunks they did not obtain; b) attempt to perform a denial of service (DoS) attack against the server or selected clients (this attack would involve only protocol messages, as we consider bandwidth or connection flooding attacks outside the scope of this work); and c) upload invalid chunks aiming at disrupting the distribution of content.

We assume that a selfish or malicious client cannot interfere with the IP routing and forwarding function, and cannot corrupt messages, but it can eavesdrop messages. In addition, we assume that communication errors may occur during message transmissions.

C. Setting and Notation

Before we describe our design, we introduce the setting and notation.

We use $\langle X \rangle$ to denote the description of an entity or object, e.g., $\langle X \rangle$ denotes a client X ’s ID, while X denotes the client itself. H is a cryptographic hash function such as SHA-1, MAC is a Message Authentication Code such as HMAC [23], and i refers to a time period (epoch). By i_X we denote the epoch i at client or server X . $MAC_K[X, Y]$ denotes the MAC of the concatenation of items X and Y , using the key K .

Each user applies for a Dandelion account and is associated with a persistent ID. The server S associates each client with its authentication information (client ID and password), the content item it currently downloads or seeds, its credit balance, and the content it can access. The clients and the server maintain loosely synchronized clocks using standard techniques, such as the Network Time Protocol (NTP).

Every client A that wishes to join the network must establish a transport layer secure session with the server S , e.g., using TLS [24]. A client sends its ID and password over the secure channel. The server S generates a random secret key, denoted K_{SA} , which is shared with A . K_{SA} is also sent over the secure channel. In addition, every Dandelion client A obtains from the server a public/secret key pair that is issued by the content provider. A ’s peers obtain the public key certificate signed by the server directly from A . K_{SA} and the public key pair are renewed upon epoch change.

The rest of the messages that are exchanged between the server and the clients are sent over an insecure channel (e.g., over plain TCP), which must originate from the same IP as

the secure session. Similarly, all messages between clients are sent over an insecure channel.

Each client A exchanges only short messages with the server. To prevent forgery of the message source and replay attacks, and to ensure the integrity of the message, each message includes a sequence number and a digital signature. The signature is computed as the MAC of the message and the sequence number, keyed with the secret key K_{SA} that A shares with the server. Each time a client or the server receive a message from each other, they check whether the sequence number succeeds the sequence number of the previously received message and whether the MAC-generated signature verifies. If either of the two conditions is not satisfied, the message is discarded. The sequence number is reset when time period i changes.

IV. Design

In this section, we describe the design of Dandelion, which explicitly addresses the challenges posed by selfish and malicious clients, as well as the communication channel.

We introduce our credit-based exchange cryptographic protocol for the fair and non-repudiable exchange of content uploads for real monetary value. We also describe our hybrid incentive scheme, which combines the credit-based exchange with the TFT-based exchange.

A. Credit as Incentives

We aim at providing strong incentives for a selfish client to upload to a peer that does not possess content of interest or to a peer that is unable to upload as fast as the client uploads to it. To this end, we employ a cryptographic protocol to ensure the fair-exchange of content uploads for credit.

This protocol involves only efficient symmetric cryptographic operations. The *server* acts as the trusted third party (TTP) mediating the exchanges of content for credit among its clients, and as a credit bank maintaining records of the clients' credit balances. When a client A uploads to a client B , it sends encrypted content to client B . To decrypt, B must request the decryption key from the server. The requests for keys serve as the proof that A has uploaded some content to B . Thus, when the server receives a key request, it credits A for uploading content to B , and charges B for downloading content.

When a client A sends invalid content to a client B , B can determine its validity only after receiving the decryption key and being charged. To address this problem, our design includes a non-repudiable complaint mechanism. If A intentionally sends garbage to B , A cannot deny that it did. In addition, B is prevented from falsely claiming that A has sent it garbage.

The server and the credit base are logical modules and can be distributed over a cluster (e.g., using consistent hashing based on client ID) to improve scalability and fault-tolerance.

B. Credit Management

Dandelion's incentive mechanism creates a market, which enables a variety of application scenarios. Our protocol is intended for the case in which users maintain paid accounts with the content provider. The currency employed by Dandelion is directly mapped to real monetary value that customers

introduce in the market by purchasing content. We employ real instead of virtual currency to eliminate depletion, inflation and starvation issues that plague typical virtual currency systems [25].

Selfish clients may sell upload service to peers that are unable to reciprocate with equally fast uploads. The content provider rewards uploaders with a credit value $\Delta_r > 0$ for the uploading of a chunk, which is fixed for every chunk and every client. Downloaders spend Δ_c credit units for each chunk they download. A client is awarded sufficient initial credit to download the complete paid content from its peers, without having to upload. In this way, slow uploaders do not face starvation and they are able to expend their credit at the rate needed to achieve their desired download rate.

The content provider redeems a client's accumulated credit for monetary rewards, such as discounts on content prices or service membership fees. We assume that the content provider prices chunk uploads appropriately to ensure that for the vast majority of clients, utility increases when they utilize their uplink in exchange for credit. We set $\Delta_r = \Delta_c$, so that two colluding clients cannot increase the sum of their credit by falsely claiming that they upload to each other. A client can acquire a chunk from a peer that is not interested in the client's content only if the client's credit is greater than Δ_c .

A user cannot boost its credit by presenting multiple IDs (the Sybil attack [26]) and claiming to have uploaded to some of its registered IDs. This is because each user maintains an authenticated paid account with the provider. The user essentially purchases its initial credit, and the net sum in an upload-download transaction between any two IDs is zero.

C. Client Access Control

Before we present Dandelion's fair-exchange mechanisms, we describe how Dandelion enables the server and its clients to determine which clients are authorized participants. We also describe how clients obtain information about the content and the swarm. Figure 1 provides a high-level description of the client access control protocol and we describe it in detail below.

Step 1: The protocol starts with the client B sending a request for the content item F to the server S .

$$B \longrightarrow S: [\text{content request}] \langle F \rangle$$

Step 2: If B has access to F , the server chooses a short list of peers $\langle A \rangle_{\text{list}}$, among the ones that are currently in the swarm for F . The policy with which these peers are selected depends on the specifics of the content distribution system. Each list entry contains the ID of the peer and the peer's inbound Internet address. For every peer A in A_{list} , S sends a ticket $T_{SA} = \text{MAC}_{K_{SA}}[\langle A \rangle, \langle B \rangle, \langle F \rangle, t]$ to B , where t is the current timestamp. The ticket T_{SA} is only valid for a certain time length L_{peer} and allows B to request chunks of the content $\langle F \rangle$ from client A . When T_{SA} expires and B still wishes to download from A , B requests a new T_{SA} from S . The ticket T_{SA} enables A to filter out service requests from misbehaving or unauthorized peers. To ensure integrity in the case of static

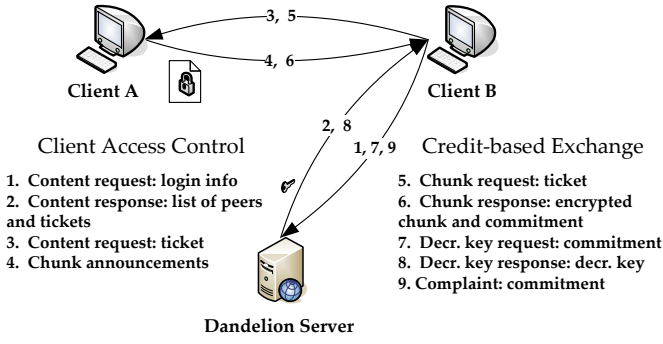


Figure 1: The client access control and credit-based exchange protocols. Messages are listed in a “message type:contents” format and only the most important contents are included. The numbers on the arrows correspond to the listed protocol messages and the steps listed in Sections IV-C and IV-D.

content or video on demand, S also sends to B the SHA-1 hash $H(c)$ for all chunks c of $\langle F \rangle$.

$$S \longrightarrow B: [\text{content response}] T_{SA_{\text{list}}}, \langle A \rangle_{\text{list}}, H(c)_{\text{list}}, \langle F \rangle, t, i_S$$

Step 3: Upon receiving the server’s response, B connects to each client $A \in A_{\text{list}}$ to request the content $\langle F \rangle$. In the rest of this description, we list only the steps that involve B , and a specific client A .

$$B \longrightarrow A: [\text{content request}] T_{SA}, \langle F \rangle, t, i_S$$

Step 4: If $\text{current-time} \leq t + L_{\text{peer}}$ and T_{SA} is not in A ’s cache, A verifies whether $T_{SA} = \text{MAC}_{K_{SA}}[\langle A \rangle, \langle B \rangle, \langle F \rangle, t]$. If the verification fails, A drops this request. Also, if i_S is greater than A ’s current time period i_A , A learns that it should renew its key with S . Otherwise, A caches T_{SA} and periodically sends the chunk announcement message described below, for the period that the timestamp t is fresh. This message contains a list of chunks that A owns, $\langle c \rangle_{\text{list}}$. B also does so in separate chunk announcement messages. The specifics of which chunks are announced and how frequently depend on the type of content distribution.

$$A \longrightarrow B: [\text{chunk announcement}] \langle c \rangle_{\text{list}}$$

D. Exchanging Content Uploads for Credit

We now describe in detail Dandelion’s cryptographic credit-based exchange protocol (Figure 1).

Step 5: B and A determine which chunks to download from each other according to a chunk selection policy. For example, BitTorrent’s locally-rarest-first [1] is suitable for static content distribution. B sends a request for the missing chunk c to A .

$$B \longrightarrow A: [\text{chunk request}] T_{SA}, \langle F \rangle, \langle c \rangle, t, i_S$$

Step 6: B ’s chunk requests are served by A as long as the timestamp t is fresh, and T_{SA} is cached or verifies. A encrypts c using a symmetric encryption algorithm Enc , as $C = \text{Enc}_{k_{(c)}}(c)$. $k_{(c)}$ is a key and encryption initialization vector pair generated as $(\text{MAC}_{K_{SA}}[\langle A \rangle, \langle B \rangle, \langle F \rangle, \langle c \rangle, t, 0], \text{MAC}_{K_{SA}}[\langle A \rangle, \langle B \rangle, \langle F \rangle, \langle c \rangle, t, 1])$. Next, A hashes the ciphertext

C as $H(C)$. Subsequently, it computes its commitment to the encrypted chunk as $T_{AS} = \text{MAC}_{K_{SA}}[\langle A \rangle, \langle B \rangle, \langle F \rangle, \langle c \rangle, H(C), t]$. The commitment T_{AS} is only valid for a certain time length L_{key} , which forces B to purchase the chunk at the server before T_{AS} expires. This fact allows A to promptly acquire credit for its service.

$$A \longrightarrow B: [\text{chunk response}] T_{AS}, \langle F \rangle, \langle c \rangle, C, t, i_A$$

Step 7: Since B does not know the key K_{SA} that was used to generate $k_{(c)}$ in step (6), it needs to request $k_{(c)}$ from the server. As soon as B receives the encrypted chunk, B computes its own hash over the received ciphertext C' and sends a decryption key request message to S .

$$B \longrightarrow S: [\text{decryption key request}] \langle A \rangle, \langle F \rangle, \langle c \rangle, H(C'), t, T_{AS}, i_A$$

Step 8: If $\text{current-time} \leq t + L_{\text{key}}$, and the reported epoch of A is off by at most one, S checks if $T_{AS} = \text{MAC}_{K_{SA}}[\langle A \rangle, \langle B \rangle, \langle F \rangle, \langle c \rangle, H(C'), t]$. The commitment’s T_{AS} verification may fail either because $C' \neq C$ due to transmission error in step (6) or because A or B are misbehaving. Since S is unable to determine which is the case, it punishes neither A or B and does not update their credit. S does not send the decryption key to B but it notifies B of the discrepancy. In case A repeatedly sends invalid chunk response messages, B is expected to disconnect from A and blacklist it. If B keeps sending invalid decryption key requests that involve A , S penalizes B . If the verification succeeds, S checks whether B has sufficient credit to purchase the chunk c . It also checks again whether B has access to the content F . If B is approved, S charges B and rewards A with Δ_c credit units. Subsequently, S computes $k'_{(c)}$ as $(\text{MAC}_{K_{SA}}[\langle A \rangle, \langle B \rangle, \langle F \rangle, \langle c \rangle, t, 0], \text{MAC}_{K_{SA}}[\langle A \rangle, \langle B \rangle, \langle F \rangle, \langle c \rangle, t, 1])$ and sends it to B .

$$S \longrightarrow B: [\text{decryption key response}] \langle A \rangle, \langle F \rangle, \langle c \rangle, k'_{(c)}$$

B uses $k'_{(c)}$ to decrypt the chunk as $c' = \text{Dec}_{k'_{(c)}}(C')$. Next, we explain the complaint mechanism.

Step 9: If the decryption fails or if $H(C') \neq H(C)$ (step (2), § IV-C), B complains to S by sending the following message.

$$B \longrightarrow S: [\text{complaint}] \langle A \rangle, \langle F \rangle, \langle c \rangle, T_{AS}, H(C'), t, i_A$$

S ignores this message if $\text{current-time} > t + L'_{\text{key}}$, where $L'_{\text{key}} > L_{\text{key}}$. $L'_{\text{key}} - L_{\text{key}}$ should be greater than the time needed for B to request and receive a decryption key response, decrypt the chunk and send a complaint to the server. With this condition, a misbehaving client A cannot avoid unfavorable complaint resolution by ensuring that the time elapsed between the moment A commits to the encrypted chunk and the moment the encrypted chunk is received by B is slightly less than L_{key} . S also ignores the complaint message if a complaint for the same A and c is in a cache of recent complaints that S maintains for each client B . Complaints are evicted from this cache once $\text{current-time} > t + L'_{\text{key}}$.

If $T_{AS} \neq \text{MAC}_{K_{SA}}[\langle A \rangle, \langle B \rangle, \langle F \rangle, \langle c \rangle, H(C'), t]$, S punishes B . This is because S has already notified B that T_{AS} is invalid in step (8). If T_{AS} verifies, S caches this complaint, re-computes $k'_{(c)}$ once again, retrieves c from its storage, and encrypts c

himself using $k'_{(c)}$, $C'' = Enc_{k'_{(c)}}(c)$. If the hash of the ciphertext $H(C'')$ is equal to the value $H(C')$ that B sent to S , S decides that A has acted correctly and B 's complaint is unjustified. Subsequently, S drops the complaint request and blacklists B . It also notifies A , which disconnects from B and blacklists it. Otherwise, if $H(C'') \neq H(C')$, S decides that B was cheated by A , blacklists A , revokes the corresponding credit charge on B and notifies B that its complaint has been resolved. Similarly, B disconnects from A and locally blacklists it.

The server disconnects from a blacklisted client A , marks it as blacklisted in the credit file and denies access to A if it attempts to login. Future complaints that concern A and are non-duplicate, non-expired and with valid commitments, are ruled against A without further processing.

E. Rate-based Tit-for-Tat with Optimistic Fair-Exchange

Below we describe how Dandelion combines the credit-based exchange and the TFT-based exchange into one hybrid incentive scheme. With the credit-based exchange, clients that cannot upload at the rate their peers upload to them, pay the excess offered bandwidth in credit. However, when two clients are mutually interested in each other's content and are able to upload to each other at the same rate, they may use tit-for-tat incentives, i.e., employ the TFT-based exchange. The TFT-based exchange mechanism aims at reducing the decryption key request load on the server. It enables a pair of clients to barter their uplink resources without requiring the server to mediate their transactions.

The *trade surplus* of a client with respect to a peer is the difference between the number of TFT-traded chunks the client has uploaded to the peer and the number of TFT-traded chunks the client has downloaded from the peer. The trade surplus threshold ts is the maximum value that the trade surplus can take before the client with the positive trade surplus switches to credit-based exchange.

If a client is a seeder, it always uploads using the credit-based exchange. Otherwise, if the client's trade surplus with a peer is less than or equal to the specified threshold, each client responds to requests for chunks from its peers with TFT-traded chunks. If the surplus exceeds ts , the client responds with credit-traded chunks.

A strawman approach for TFT-based exchange would involve employing a tit-for-tat scheme under which clients exchange plaintext chunks, as long as the trade surplus does not exceed a threshold [5]. However, this scheme cannot guarantee absolute fairness, as the last peer that receives content may refrain from reciprocating. If in addition a free-rider employs the "large view exploit" or the trade surplus threshold is high, a free-rider can download a substantial amount of content without incurring any cost (see § VI-C). The free-rider problem is exacerbated under Dandelion, as the gains of free-riders translate to monetary losses for their peers.

To address this problem, we employ an optimistic fair-exchange scheme, which allows clients to barter their uplink bandwidth in a fair manner. This scheme involves the server only in case of client misbehavior. It is adapted from BAR

Gossip [11] and classic optimistic fair-exchange protocols [12, 13]. Next, we provide a high-level description of the cryptographic TFT-based exchange protocol. Due to space limitations and its similarity to previously proposed schemes we omit a detailed description and refer interested readers to [27].

When a client A uploads a TFT-traded chunk c_1 to a client p , c_1 is encrypted using symmetric key cryptography. To decrypt, p must reciprocate with at least one TFT-traded chunk c_2 to A . After receiving c_2 , A sends the decryption key for c_1 to p . In turn, p reciprocates with the decryption key for c_2 . A sends TFT-traded chunks to p as long as the trade surplus does not exceed the threshold.

Every client maintains the decryption key surplus dks_p for each of its peers p . dks_p is the difference between the number of decryption keys for TFT-traded chunks the client sent to p and the number of decryption keys the client received from p . A client sends to p a decryption key for a sent TFT-traded chunk if both of the following two conditions are satisfied: a) $dks_p < 1$; and b) it has received TFT-traded chunks from p for which it has not received decryption keys or $dks_p = -1$.

If p sends an invalid chunk c_2 to A , A can detect it only after sending the decryption key for the valid chunk c_1 to p . To address this issue, we include a non-repudiable complaint mechanism similar to the one used by the credit-based exchange. Unlike the credit-based exchange however, in the TFT-based exchange, senders commit to chunks and decryption keys they send using public key signatures. This is because the server is involved only in case of client misbehavior. Therefore, clients should be able to determine the validity of the commitments to the transmitted chunks without querying the server as in step (7) (§ IV-D).

A Dandelion client must be able to switch between the credit- and TFT-based exchange depending on content availability and peer upload and download rates. Therefore, we need an algorithm that aims at reducing the amount of credit-based uploads to each peer, while it ensures that the client uploads to its peers at the maximum rate its peers can download from it.

Rather than employing a complex per-peer resource allocation algorithm, we use the following simple scheme. At any moment, a client selects a specified number n of peers to which to upload using a downloader selection algorithm almost identical to BitTorrent's. The only difference with BitTorrent is that a Dandelion leecher ranks its peers based only on the rate with which they upload TFT-traded chunks to the leecher.

When a leecher selects the fastest TFT-traded chunk uploaders, it selects peers that are more likely to match its own TFT-based uploads. This results in invoking the credit-based exchange less frequently.

F. Design Properties

We now list the properties of our design. For brevity of presentation, we omit proofs on why these properties hold. They can be found in [27].

Lemma 1 A selfish or a malicious client cannot assume another authorized client A 's identity and issue messages under

A. Thus, it cannot obtain service at the expense of *A* or cause *A* to be charged for service it did not obtain or cause *A* to be blacklisted. In addition, it cannot issue a valid T_{SA} for an invalid chunk that it sends to a client *B* and cause *B* to produce a complaint message that would result in a verdict against *A*.

Lemma 2 If the server *S* charges a client *B* Δ_c credit units for a chunk *c* received from a selfish client *A*, *B* must have received the correct *c*, regardless of the actions taken by *A*.

Lemma 3 If a selfish client *A* always encrypts chunk *c* anew when serving a request, as described in step (6) (§ IV-D), and if *A* sends a valid *c* to *B*, then *A* is awarded Δ_c credit units from *S*, and *B* is charged Δ_c credit units from *S*.

Lemma 4 A malicious client cannot replay previously sent valid requests to the server or generate decryption key requests or complaints under another client *A*'s ID. Thus, it cannot cause *A* to be charged for service it did not obtain or cause *A* to be blacklisted because of invalid or duplicate complaints.

Lemma 5 Under the TFT-based exchange scheme, a selfish client *B* cannot obtain a chunk c_1 from its peer *A* without expending bandwidth to reciprocate with a valid chunk c_2 itself. In addition, as stated in [11], a rational *B* prefers to send the short decryption key for an already sent valid chunk c_2 , rather than repeatedly receive requests for the key from *A*.

Observation 1 To maintain an efficient content distribution pipeline, a client needs to relay a chunk to its peers as soon as it receives it. However, the chunk may be invalid due to communication error or due to client misbehavior. The performance of the system would be severely degraded if clients wasted bandwidth to relay invalid content. To address this issue, Dandelion clients send a decryption key request to the server immediately upon receiving the encrypted chunk. This design choice enables clients to promptly retrieve the chunk in its non-encrypted form and verify its integrity prior to uploading the chunk to their peers.

Observation 2 If a client does not have sufficient credit, it cannot download chunks from a selfish peer that is not interested in the client's content. Our design choice to involve the server in each exchange of content uploads for credit enables the server to check a client's credit balance, before the client retrieves the decryption key of a chunk.

Observation 3 A malicious client *B* can abandon the credit-based exchange protocol after receiving the encrypted chunk without completing the transaction. In such case, *A* does not receive any credit, even though *B* has consumed *A*'s resources. This is a denial of service (DoS) attack against *A*. Note that this attack would require client *B* to expend resources proportional to the resources of the victim *A*, therefore it is not particularly practical. Furthermore, we prevent clients that have been designated as misbehavers (blacklisted) in step (9) or clients that do not maintain paid accounts with the content provider from launching such attacks; the server *S* issues short-lived tickets T_{SA} (step (2), § IV-C) only to authorized and non-blacklisted clients.

Observation 4 A malicious client *A* may send a credit-traded chunk with an invalid MAC signature aiming at performing a DoS attack against *B*, without becoming blacklisted by the server. This attack would require client *A* to expend resources

proportional to the resources of the victim *B*, therefore it is not particularly practical. In addition, a victim can be attacked by only one chunk before it locally blacklists the attacker. Furthermore as before, we prevent unauthorized clients or ones that have been blacklisted by the server from launching such attack.

Observation 5 A malicious client cannot DoS attack the server by sending invalid content to other clients or repeatedly sending invalid complaints aiming at causing the server to perform complaint resolution. That client must be a user registered with the system, otherwise it is not able to mint a complaint that merits resolution. Even if the client is a registered user, it becomes blacklisted by both the server and its peers the moment an invalid complaint is ruled against it. In addition, a malicious client cannot attack the server by sending valid signed messages with duplicate valid complaints. Our protocol detects duplicate complaints through the use of timestamps and caching of recent complaints.

Owing to Lemmas 1, 2, 3 and 5 as well as Observation 2, and given that the content provider appropriately values chunk uploads, Dandelion ensures that most selfish clients increase their utility when they upload correct chunks. At the same time, misbehaving clients cannot increase their utility. Consequently, Dandelion provides strong incentives for most selfish clients to upload to their peers.

G. Discussion

We now discuss our scheme's economic viability and potential for adoption. We argue that a content provider obtains more gains using our approach than by using a protocol such as BitTorrent that does not provide robust incentives for seeding.

When seeding is not strongly incentivized, a content provider needs to purchase additional hardware and bandwidth to directly provide a large portion of the required upload capacity. On the other hand, Dandelion enables the content provider to make a less expensive investment towards rewarding cooperative peers with real money. As a result, peers are strongly incentivized and the total upload capacity of the swarm increases.

Next, we support our insight that it is cheaper for content providers to purchase bandwidth from their users than purchase the infrastructure to directly serve content or purchase the service of third party CDNs. We make the conservative assumption that half of the price paid by broadband customers goes towards purchasing the uplink bandwidth. Based on current DSL, Cable and FiOS offers in the US we extrapolate that user uplink bandwidth costs between \$2 and \$5 per Mbps per month [28, 29]. On the other hand, depending on location, it costs at least \$40 to \$80 per Mbps for a content provider to purchase T-3 to OC-12 bandwidth, with the cost of an OC-12 installation being on the order of \$500000 [30].

Therefore, although with Dandelion the content provider expends money to purchase its clients' bandwidth, he might incur lower cost compared to purchasing server bandwidth from Internet service providers. Although a user's uplink bandwidth may cost more than the content provider is willing to pay, that bandwidth is typically unused or altruistically

assigned to other P2P applications. Therefore, we hypothesize that our scheme can enable users to benefit from their spare bandwidth and content providers to tap into that relatively low cost resource. Validating this hypothesis requires a real market experiment, which is beyond the scope of this work.

V. Implementation

This section describes a prototype C implementation of the Dandelion system, which is suitable for static content distribution.

A. Server Implementation

For simplicity, our current implementation combines the content provider and the credit management system at a single server. It is our future work to scale the Dandelion server by balancing its load over multiple machines.

Our current server implementation is single-threaded and event-driven. The network I/O operations are asynchronous, and data are transmitted over TCP. In order to scale to thousands of simultaneously connected clients, the server employs the `epoll` event dispatching mechanism.

The server uses standard file I/O system calls to efficiently manage persistent client information, which is stored in a simple file called the credit file. Each client is assigned an entry in the credit file, which keeps the client’s credit, its authentication information and its file access control information. Each entry has the same size and the client ID determines the offset of the entry of each client in the file. Thus each entry can be efficiently accessed for both queries and updates.

The server queries and updates a client’s credit from and to the credit file upon every transaction. Yet, it does not force commitment of the update to persistent storage. Instead, it relies on the OS to asynchronously perform the commitment.

B. Client Implementation

The client side is also single-threaded and event-driven. A client may leech or seed multiple files at a time. A client can be decomposed into two logical modules: a) the *connection management* module; and b) the *peer-serving* module.

The connection management module performs *peering* and *uploader discovery*. With peering, each client obtains a random partial swarm view from the server and strives to connect to a specified number of peers (typically 50-100). With uploader discovery, a client strives to remain connected to a minimum number of uploading peers.

The peer-serving module performs *content reconciliation* and *downloader selection*. Content reconciliation refers to the function of announcing recently received chunks, requesting missing chunks, requesting decryption keys for received encrypted chunks, and replying to chunk requests. Our implementation employs rarest-random-first [31] scheduling in requesting missing chunks from clients. To efficiently utilize their downlink bandwidth, clients dynamically adjust the number of outstanding chunk requests that a client has sent to a peer and has not received a response for. We described the downloader selection algorithm in § IV-E.

VI. Evaluation

The goals of this experimental evaluation are: a) to identify the scalability limits of Dandelions centralized nonmanipulable virtual-currency; b) to examine the trade-off between performance and scalability in selecting the parameters of Dandelions hybrid incentive scheme; c) to motivate our design by demonstrating the importance of incentives for seeding and the impact of free-riding in BitTorrent-like swarms.

A. Server Performance

In this section, we evaluate and profile the server in terms of decryption key and complaint request throughput.

1) Server Throughput

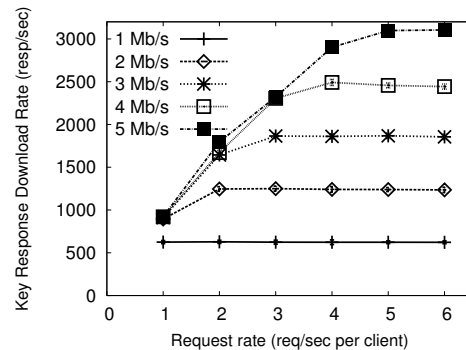


Figure 2: Aggregate server decryption key response throughput as a function of specified per-client key request rate. We use ~ 1000 clients and we vary the server rate-limits.

A Dandelion server mediates the chunk exchanges between its clients. The client plaintext download throughput and the scalability of our system is bound by how fast a server can process their decryption key requests (step (8), § IV-D). Both the server’s computational resources and bandwidth may become the performance bottleneck. We deploy a Dandelion server on a dual Pentium D 2.8GHZ/1MB CPU with 1GB RAM and 250GB/7200RPM HDD running Linux 2.6.5-1.358smp, which shares a 100Mbps Ethernet II link. To mitigate bandwidth variability in the shared link and to emulate a low cost server with uplinks and downlinks that range from 1Mbps to 5Mbps, we rate-limit our Dandelion server at the application layer. We deploy ~ 1000 clients that run on ~ 100 distinct PlanetLab hosts.

The clients send requests for decryption keys to the server and we measure the aggregate rate with which all clients receive decryption key responses. The server always queries and updates the credit record from and to the credit file without forcing commitment to disk. We run each experiment for a specified per-client request rate, which varies from 1 to 6 req/sec. For each request rate, the experiment duration is 10 minutes and the results are averaged over 10 runs.

Figure 2 depicts the server’s decryption key response throughput for varying server bandwidth. As the bandwidth increases from 1Mbps to 4Mbps, so does the server’s throughput, indicating that for up to 4Mbps access link, the bottleneck is the bandwidth. For 5Mbps and 4Mbps the throughput is

almost equal, indicating that for 5Mbps the bottleneck is the CPU. The results show that a server running on our commodity PC with 4Mbps or 5Mbps access link can process up to ~ 3105 decryption key requests per second. This result suggests that with a 256KB chunk size, this server may simultaneously support almost 3100 clients that download only credit-traded chunks at 256KB/s. With a larger chunk size and TFT-based exchange, each such client receives credit-traded chunks at a lower rate. Thus, the number of supported clients increases.

We also compare the throughput of the 5Mbps server in case ~ 500 clients send 10 req/sec each (3114 req/sec, not depicted in Figure 2) with its throughput when ~ 1000 clients send 5 req/sec each (3105 req/sec). This result suggests that the throughput is mostly independent of the number of clients due to the scalability of the `epoll` event dispatching mechanism.

A Dandelion server is also responsible for resolving complaints (step (9), § IV-D). A complaint resolution involves the expensive disk I/O operation for reading a chunk. Therefore it represents a performance bottleneck in case the system receives too many complaints. We performed an experiment with ~ 1000 clients sending 5 decryption key req/sec, and 15 clients sending 1 complaint resolution request per sec (involving a randomly selected 256KB chunk of a 1GB file). The 5Mbps server was able to deliver roughly 1490 decryption key responses per second along with 14 complaint resolution responses per second.

Note that the server does not need to deliver high complaint resolution throughput for the reasons listed in Observation 5, § IV-F. In addition, to improve throughput, the expensive disk I/O operation can be performed in parallel with the decryption key request processing using asynchronous I/O.

2) Server Profiling

We profile the cost of operations at the server aiming at identifying the performance bottlenecks of our design. We use the same machine as the one used in the previous section.

Table I lists the cost of Dandelion operations. Timings for operations 1-4 and 6-8 are obtained using `getrusage()` over 10000 executions. Timings for operations 5, 12 and 13 are approximated using `gettimeofday()` over 10000 executions. Operation 5 reads from the disk a new randomly selected 256KB chunk of a 1GB file in each execution. Operations 12-13 are performed on a credit file with 10000 44-byte entries. Timings for operations 9-11 are approximated according to our application layer rate-limiting for 5Mbps uplink and downlink. They are provided as reference for comparison with CPU-centric and credit management operations. Operation 6 uses 8-byte-block Blowfish-CBC with 128-bit key and 128-bit initialization vector. Operations 1-4 use HMAC-SHA1 with 128-bit key. Operation 7 uses SHA-1. Operation 8 uses 1024-bit RSA signatures.

The main tasks of a Dandelion server are to: a) receive the decryption key request (operation 9); b) authenticate the decryption key request (operation 1); c) verify the commitment (operation 2); d) compute the decryption key (operation 3); e) query and update the credit of the two clients involved (operations 12 and 13); f) sign the decryption key response (operation 4); and g) send the decryption key response (operation 10).

	Dandelion operation	Size	Time (ms)
CPU-centric Operation			
1	Authenticate decryption key request	58 bytes	.003
2	Generate symmetric cryptography commitment for decryption key request or complaint verification	38 bytes	.003
3	Compute decryption key	19 bytes	.003
4	Sign decryption key response	46 bytes	.003
5	Read chunk	256 KB	31
6	Encrypt chunk	256 KB	2.876
7	Hash encrypted chunk	256 KB	1.017
8	Verify public-key commitment for complaint verification	128 bytes	.2
Communication Operation			
9	Receive decryption key request	96 bytes	$\sim .26$
10	Transmit decryption key response	84 bytes	$\sim .24$
11	Receive TFT-based exchange complaint	204 bytes	$\sim .55$
Credit Management Operation			
12	Query credit file	N/A	~ 0.004
13	Update credit file without commit to disk (rely on OS)	N/A	~ 0.02

Table I: Timings of Dandelion operations.

The signed decryption key request and decryption key responses are sent over an insecure TCP connection. A client establishes and uses the secure TLS channel with the server only to send authentication information (once per Dandelion session), the shared key and the public key pair (the same keys are used for a relatively long period).

As can be seen in Table I, the per-decryption-key-request cryptographic operations of the server (operations 1-4) are highly efficient (total 12 μ sec), as only symmetric cryptography is employed. The credit management operations (12 and 13) are also efficient (total 24 μ sec). The communication costs of receiving and sending decryption key responses (operations 9-10) are clearly higher than the cryptographic computation costs. In addition, operations 9-10 can take place concurrently with each other and the computational operations.

The cost of a complaint is substantially higher because in addition to receiving the message (operations 9 or 11), authenticating it and verifying a commitment (operations 2 or 8), it involves reading a chunk (operation 5), encrypting it with the sender's key (operation 6), and hashing the encrypted chunk (operation 7).

B. System Performance

In this section, we experimentally evaluate the behavior of the entire Dandelion system on PlanetLab. We examine the impact of chunk size and the trade surplus threshold on the performance of the system. We also quantify the scalability gains obtained through Dandelions new hybrid scheme. In addition, we demonstrate the performance gains of providing incentives for seeding. In all experiments we run a Dandelion server on the same machine as the one used in the previous sections, and the server is rate-limited at 5Mbps.

Leechers are given sufficient initial credit to completely download a file, according to the credit management policy discussed in § IV-B. Clients always respond to chunk requests from their selected downloaders.

We aim at making our evaluation representative of real Internet peer-to-peer content distribution swarms, while including

Portion of nodes	0.05	0.05	0.1	0.1	0.1	0.1
Bandwidth (KB/sec)	40	50	55	60	65	70
Portion of nodes	0.05	0.1	0.05	0.05	0.05	0.2
Bandwidth (KB/sec)	75	100	150	200	250	350

Table II: Distribution of upload bandwidth of Dandelion and BitTorrent peers as used in our PlanetLab experiments. This distribution draws from the one reported in [32], but due to PlanetLab bandwidth constraints we omit hosts with upload capacity higher than 350KB/sec.

as many PlanetLab nodes as possible. To this end, we partially emulate a typical client uplink bandwidth distribution [32] (Table II) by applying per-client application layer rate-limiting.

We periodically use Dandelion to distribute a 100MB file to ~ 500 non-rate-limited hosts and we identify ~ 400 nodes that are able to attain upload rates equal to or higher than 350KB/sec. The 350KB/sec upload cap is plausible because we expect Dandelion clients to reside behind privately owned residential broadband links. These links are reported to currently offer at most ~ 3 Mbps upload capacity [33]. In addition, we impose a download rate distribution to approximate the effect of asymmetric broadband links. Clients with less than or equal to 70KB/sec upload rate, are assigned a maximum download rate that is 5 times higher than their maximum upload rate. The rest of the nodes are assigned a download rate equal to 350KB/sec.

For each configuration we repeat the experiment 10 times and we extract mean values and 95% confidence intervals over the swarm-wide *download completion times*.

We note that we evaluate a particular implementation of Dandelion that is suitable for static content distribution. Although our results would vary for other P2P content distribution applications that use different chunk scheduling and peer selection policies, we expect our results to be qualitatively similar. In particular we expect seeding to be beneficial and the chunk size and trade surplus threshold to affect performance, regardless of the specifics of the content distribution system.

1) Selecting Chunk Size and Trade Surplus Threshold

With this series of experiments we examine the trade-offs involved in selecting the size of the chunk and the trade surplus threshold of the TFT-based exchange. In addition, we motivate our hybrid incentive mechanism by quantifying its improvement in scalability over the credit-based-exchangeonly scheme proposed in [9].

Intuitively, since clients are able to serve a chunk only as soon as they obtain it, a smaller chunk size yields a more efficient distribution pipeline. In addition, when the file is divided into many pieces, chunk scheduling techniques such as rarest-first can be more effective; clients can promptly discover and download content of interest. However, a smaller chunk size increases the rate with which key requests are sent to the server, reducing the scalability of the system. Also, due to TCP’s slow start, a small chunk size cannot ensure high bandwidth utilization during the TCP transfer of any chunk. Last, small chunks yield increased control overhead.

In addition, under our optimistic fair exchange scheme, a receiver is able to acquire a TFT-traded chunk only after it reciprocates with a chunk of equal size and retrieves the

decryption key. The larger a received TFT-traded chunk is, the longer the receiver may have to wait until it is able to respond with an equally large chunk. Only after decrypting the chunk the receiver is able to relay it to its peers, therefore a large chunk decreases the efficiency of the distribution pipeline.

As the trade surplus threshold ts and the chunk size increases, trading flexibility also increases. This enables a client to upload TFT-traded chunks in case its peers cannot temporarily match the client’s upload rate. This results in reduction of the rate with which decryption key requests are sent to the server. However, a large threshold and chunk size results in clients wasting bandwidth to transmit encrypted chunks that are never reciprocated and decrypted, causing performance degradation.

We use as performance metrics the mean download completion time of the clients and the decryption key request load on the server. In each configuration, we deploy approximately ~ 400 Dandelion leechers and one initial seeder. Leechers start downloading the file almost simultaneously emulating a flashcrowd. The duration of each experiment is 2200 sec.

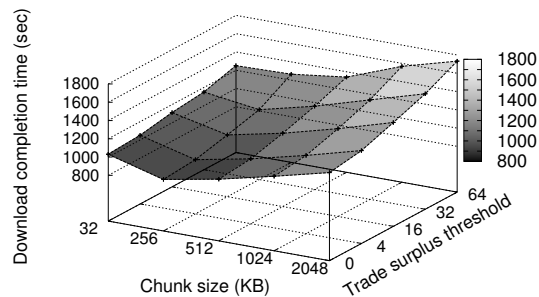


Figure 3: Swarm-wide mean download completion times of ~ 400 leechers as a function of chunk size and trade surplus threshold ts for a 100MB file. Both the z axis and the gray map depict the download completion time. The 95% confidence intervals (not depicted) take values between 50 and 100 sec.

Figure 3 shows the leecher mean download completion time as a function of the chunk size and the trade surplus threshold. We observe that for larger than 256KB chunks, the system’s performance degrades as the chunk size and the trade surplus threshold increases. For example, for $ts = 0$, a 256KB chunk size yields better performance (864 sec) than a 2048KB chunk size (1263 sec). 256KB chunks guarantee that there are sufficiently many distinct chunks for peers to exchange. The beneficial impact of smaller than 256KB chunks in terms of chunk scheduling flexibility is negated by the performance-degrading TCP effects and the increased control overhead. For example, for $ts = 0$, a 256KB chunk size yields notably better performance (864 sec) than a 32KB chunk size (1031 sec). In addition, we observe that the mean download completion times consistently increases with ts .

In Figure 4, we observe that the load on the server decreases as ts increases. In particular, under our network configuration the decryption key request load decreases by approximately 40% when the system uses $ts = 16$ instead of $ts = 0$. At the same time, the swarm-wide performance degrades only by 9 to

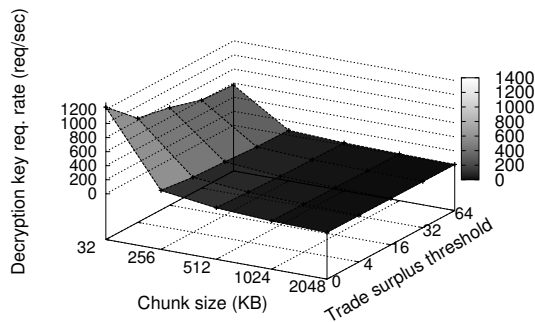


Figure 4: The mean decryption key request rate at the server as a function of the chunk size and the trade surplus threshold ts for a 100MB file. Request rates are extracted in 10 sec intervals over the duration of the experiment. The 95% confidence intervals (not depicted) take values between 3 and 30 req/sec.

13%, depending on chunk size. Setting $ts = 0$ corresponds to using only credit-based exchange as was originally proposed in [9], while $ts > 0$ allows clients that are mutually interested in each others content to exchange chunks without involving the server. This result demonstrates the effectiveness of our hybrid incentive scheme in improving scalability by reducing the servers decryption key request load.

As expected, the server load also decreases as the chunk size increases. The decryption key request load for 32KB chunks varies in ~ 600 to ~ 1200 req/sec depending on ts . For 256KB chunks it varies in only ~ 70 to ~ 190 req/sec. The evaluation for 32KB chunks enables us to roughly predict the load on the server in a swarm that consists of ~ 8 times more clients but uses 256KB chunks

For this particular swarm configuration, the content provider may determine that a 256KB chunk and $ts=4$ is a good configuration. It yields a low download completion time (893 sec) and a relatively low server load (132 req/sec). Unless mentioned otherwise, in the rest of this evaluation we use these values.

2) Impact of Seeders

Dandelions credit-based exchange mechanism strongly incentivizes clients to remain online after download completion, increasing the number of seeders in a swarm. With this series of experiments, we motivate our credit-based exchange mechanism by demonstrating the performance gains by the existence of additional seeders.

Intuitively, since typical P2P clients reside behind asymmetric links, content distribution swarms are expected to benefit by the existence of additional seeders. Seeders complement the swarms uplink bandwidth without expending its downlink bandwidth. We demonstrate the impact of seeders in BitTorrent-like swarms by varying the probability that a leecher remains online to seed a file after it completes its download. Upon completion of its download, each leecher stays in the swarm and seeds with probability a , which varies in 0% to 100%. Leechers start downloading the file immediately upon arriving in the swarm. The duration of each experiment is 2200 sec.

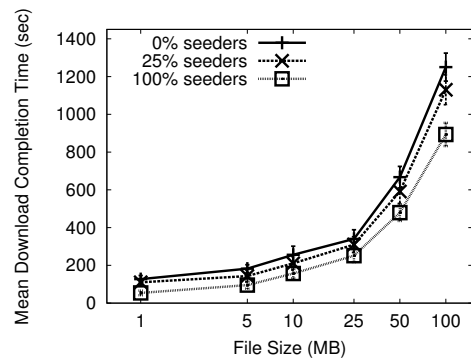


Figure 5: Swarm-wide mean download completion times of ~ 400 leechers as a function of file size for varying portion of leechers that become seeders. Clients arrive almost simultaneously.

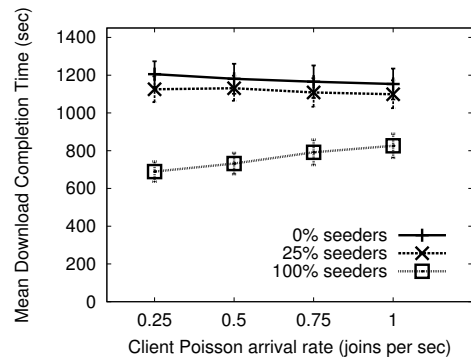


Figure 6: Swarm-wide mean download completion times of ~ 400 leechers as a function of the client Poisson arrival rate λ for varying portion of leechers that become seeders. Clients download a 100MB file.

Figure 5 depicts the mean download completion time over ~ 400 leechers as a function of the file size, for varying a . All clients join the swarm almost simultaneously. We vary the file size to demonstrate that the impact of seeding depends on the duration of the download, and to demonstrate the behavior of the system under different workloads. Our results show the beneficial impact of seeders. For example, for a 100MB file, we observe a swarm-wide mean download completion time of 893 sec and 1250 sec when $a = 100\%$ and $a = 0\%$, respectively. We observe that as the file size decreases, the decrease of a causes a more dramatic increase of download completion times. The larger the file is, the longer leechers remain online to download it, thus they upload to their peers for longer periods. For smaller files however, peers have to rely heavily on leechers that become seeders.

We also evaluate the system under varying peer arrival patterns. We vary the Poisson parameter λ under which new clients join the swarm. Depending on the arrival pattern, seeders may play a more or a less beneficial role. For example, during a flash crowd (high λ) many peers finish their download at approximately the same time and therefore do not benefit each other when they remain online as seeders. Figure 6 depicts the mean download completion time over all ~ 400 leechers as a function of the client Poisson arrival rate λ , for varying a and a 100MB file. The results show that seeders substantially benefit swarms with low arrival rates, as new

peers take advantage of the additional uplink capacity of peers that arrived earlier and became seeders. For example, for $\lambda = 0.25$, we observe a swarm-wide mean download completion time of 689 sec and 1125 sec when $a = 100\%$ and $a = 0\%$, respectively.

3) Comparison with BitTorrent

Unlike BitTorrent, Dandelion’s incentive mechanism requires the involvement of a centralized component, uses optimistic fair-exchange of content uploads, employs a modified downloader selection algorithm and does not employ subpiecing [1]. In this section, we show that these differences do not have a negative impact on download completion time.

To this end, we compare the performance of a swarm of Dandelion clients with a swarm of BitTorrent (CTorrent DNH-3.2) clients. In both swarms, there are ~ 400 leechers and one initial seeder, and leechers stay online to seed after download completion. Dandelion clients employ both the credit-based and TFT-based exchange protocols.

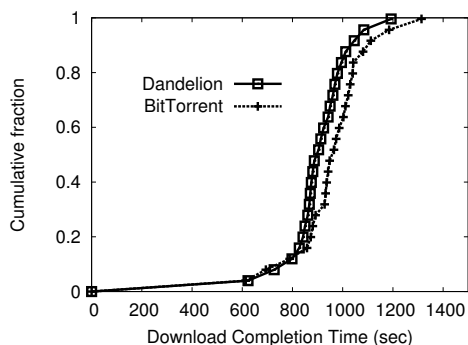


Figure 7: CDF of download completion times of ~ 400 BitTorrent and Dandelion clients that download a 100MB file. Dandelion and BitTorrent clients have average download completion time equal to 893 sec and 937 sec, respectively.

Figure 7 presents the CDF of the download completion times for both BitTorrent and Dandelion clients for a 100MB file. This illustration shows that a Dandelion swarm can attain performance comparable to a BitTorrent one, when both swarms have the same number of seeders. Although our Dandelion implementation appears to outperform BitTorrent, we do not claim that a Dandelion-based static content distribution system is better-performing. The performance of both protocols is highly dependent on numerous parameters, which we have not exhaustively analyzed.

C. Free-riding in BitTorrent-like Swarms

In this section, we provide additional motivation for the use of non-manipulable cryptographic fair-exchange incentives. We demonstrate that under BitTorrent-like incentives, free-riding is beneficial for free-riders and harmful for cooperative clients.

For all experiments we use a Dandelion implementation in which we disable the cryptographic fair-exchange protocols. Unless mentioned otherwise, we also disable the trade surplus mechanism. With disabled fair-exchange protocols, Dandelion’s implementation is almost identical to BitTorrent’s. We

use this implementation because it includes a trade-surplus mechanism and we have also validated it against BitTorrent (§ VI-B3). We deploy ~ 400 leechers and one initial seeder. All clients join the swarm simultaneously to download a 100MB file divided in 256KB chunks. The duration of each experiment is 2200 sec. Free-riders never upload, nor do they expend credit.

In each experiment, the swarm includes a group of 20 free-riders and a group of 20 cooperative clients all of which have upload and download rate-limits equal to 100KB/sec and 350KB/sec, respectively. In the rest of this evaluation we call the groups of the 20 free-rider and 20 cooperative clients, the *free-rider* and the *reference cooperative* group, respectively. The rest of the leechers are rate-limited according to the distribution used in § VI-B. Unless mentioned otherwise, cooperative and free-rider clients connect to roughly 50 and 350 peers at a time, respectively.

For each configuration we repeat the experiment 10 times and we extract mean values and 95% confidence intervals of client download rates. If the client completes its download during the experiment, its download rate is equal to the size of downloaded content divided by the download completion time. Otherwise, its download rate is the size of downloaded content divided by the experiment duration.

Figure 8(a) compares the two groups when the portion of leechers that remain online to seed varies from 0 to 100%. With this measurement we show that the “large view” exploit (§ II-B) enables free-riders to tap into scarce system resources and harm compliant clients by monopolizing the seeders and exploiting optimistic unchoking. For comparison purposes, for each percentage of leechers that become seeders we also depict the download rate of the cooperative group in the absence of free-riders.

We observe that free-riders obtain almost equal download rates with their cooperative counterparts in under-provisioned swarms with 0% to 25% seeders. Compliant clients suffer a performance hit of approximately 15%, comparing to their performance in the absence of free-riders. When the swarm has 50% to 100% leechers that become seeders, free-rider clients achieve 5% to 10% higher download rates than cooperative ones. This result confirms the potential for wide adoption of free-riding. In well-provisioned swarms, the download rate of cooperative clients degrades by roughly 10% comparing to their rate in the absence of free-riders.

In Figure 8(b), we compare the average performance of the free-rider and the 20-client reference cooperative group as the number of free-riders ranges from 0 to 100 clients. All leechers become seeders upon download completion. This measurement shows that the wide adoption of free-rider clients causes substantial performance degradation in BitTorrent swarms. When the number of free-riders varies in 50 to 100, the reference cooperative group attains approximately 20% to 30% worse performance than in a swarm with no free-riders. We also observe that as the number of free-riders increases, free-riders do not fare as well comparing to compliant clients.

Figure 8(c) depicts the performance of the free-rider group when free-riders do not download from seeders and cooperative clients employ a chunk-level TFT scheme using the

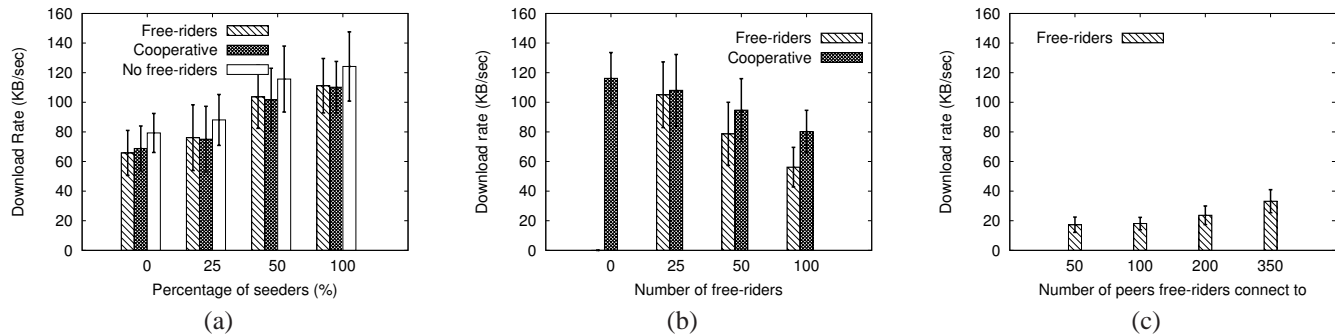


Figure 8: Swarm-wide mean download rates of a group of 20 free-riders and a group of 20 cooperative clients that join a swarm of ~ 350 leechers to download a 100MB file. (a) Download rates for varying percentage of peers that remain online seeding after download completion. We also depict the download rate of cooperative clients in the absence of free-riders (“No free-riders”); (b) Download rates for varying number of free-riders; (c) Download rates of free-riders when they cannot download from seeders, for varying number of peers that they connect to.

trade surplus mechanism. Under this scheme, leechers upload plaintext chunks to their selected downloaders as long as the trade surplus does not exceed 1. BitTorrent does not currently prevent free-riders from downloading from seeders. On the other hand, Dandelion seeders are motivated to upload only encrypted content, for which they are rewarded. No leechers become seeders. The number of peers to which free-riders connect to varies in 50 to 350 to illustrate the impact of the “large view” exploit.

As can be seen in Figure 8(c), when free-riders connect to 350 peers, they can attain up to 33KB/sec. Although this is not a good download rate by itself, recall that any gains of a Dandelion free-rider translate to monetary losses for its peers. With this measurement, we show that the credit-based exchange substantially reduces the free-rider download rates. We also motivate Dandelion’s TFT-based exchange. That is, we show that even if we employ credit-based exchange and enforce strict chunk-level tit-for-tat, free-riders that employ the “large view” exploit are able to download non-negligible amounts of content without expending credit or uplink bandwidth.

VII. Related Work

We discuss previous work on incentives for cooperation in peer-to-peer content distribution systems. Due to space limitations we provide an abbreviated discussion on existing incentive schemes and we omit previous work on cryptographic fair-exchange. For more detailed exposition, we refer the reader to [27].

In P2P content distribution protocols that employ pairwise virtual currency as incentives, e.g. [15, 34–36], clients maintain credit balances with each of their peers. In this context, credit refers to any metric of a peer’s cooperativeness. These pairwise credit-based incentive mechanisms bear weaknesses that are similar to the ones of rate-based tit-for-tat: a) they provide no explicit incentives for seeding; and b) they can be manipulated by free-riders that obtain a “large view” of the network.

BAR Gossip [11] is designed for P2P streaming of live content. Owing to its public-key-based cryptographic fair-exchange mechanism, it is robust to clients that attempt to free-ride. However, to ensure fairness, BAR Gossip clients that

receive initial optimistic offers (termed Optimistic Push) from their peers need to expend bandwidth in order to reciprocate with invalid or no longer relevant chunk transmissions. In addition, a BAR Gossip client can download only as fast as it can upload. In contrast, if Dandelion clients are not able to reciprocate, the system switches to credit-based exchange and clients may purchase the excess bandwidth. Dandelion, which needs to incentivize seeding, guarantees fair-exchange of content uploads for credit. On the other hand, since in P2P live streaming peers are concurrently interested in the same content, BAR Gossip only provides tit-for-tat incentives.

Karma [37] employs a global credit bank and fair-exchange of content for reception proofs. It distributes credit management among multiple nodes. Karma’s distributed credit management improves scalability. However, it does not guarantee the integrity of the global currency in a highly dynamic network or when the majority of the nodes comprising the credit bank are malicious. In contrast, Dandelion’s centrally maintained global currency is non-manipulable by clients, enabling a content provider to incentivize client cooperation by offering monetary rewards.

PPay [38] and more recently PACE [39] are micropayment proposals that employ public key cryptography and are designed for P2P content distribution. MNet [10] uses a combination of pairwise balances and tokens that can be cashed in a central broker. These schemes do not guarantee fair-exchange of content for payment. Free-riders may establish short-lived sessions to many peers, and download substantial amounts of content or obtain payments without paying or uploading, respectively. In addition, free-riders may send payments that do not reflect real credit value (double-spending).

VIII. Conclusion

This paper describes Dandelion: an incentive scheme for cooperative (P2P) distribution of paid content. Its primary function is to enable a content provider to motivate its clients to contribute their uplink bandwidth.

Our scheme rewards cooperative clients with credit or with reciprocal uploads from their peers. Since it employs non-manipulable cryptographic schemes for the fair exchange of resources, the content provider can redeem a client’s credit for

monetary rewards. Thus, our design provides strong incentives for clients to seed content and eliminates free-riding.

Our experimental results show that a Dandelion server running on commodity hardware and with moderate bandwidth can scale to a few thousand clients. Dandelion's deployment in medium size swarms demonstrates that seeding substantially improves swarm-wide performance and that a Dandelion-based content distribution system can attain performance comparable to BitTorrent. It also demonstrates that the proposed hybrid incentive scheme significantly reduces the load on the server when compared to our previously fully centralized incentives. These facts illustrate the plausibility of our design choice: centralizing the incentive mechanism in order to increase resource availability in P2P content distribution.

Acknowledgments

We are thankful to Eddie Kohler, Nikitas Liogkas and the anonymous reviewers for their fruitful feedback on this work. We also thank Lichun Bao, Jong Han Park, Rex Chen and Denh Sy with Calit2 for providing assistance, space and equipment for our experiments. This work was supported by NSF award CNS-0627166.

References

- [1] B. Cohen, "Incentives Build Robustness in BitTorrent," in *P2P Econ*, June 2003.
- [2] C. Huang, J. Li, and K. W. Ross, "Can Internet Video-on-Demand Be Profitable?" in *SIGCOMM*, August 2007.
- [3] "BitTorrent, Inc Launches The BitTorrent Entertainment Network," www.bittorrent.com/about/press/bittorrent-inc-launches-the-bittorrent-entertainment-network, Feb 2007.
- [4] D. Hughes, G. Coulson, and J. Walkerdine, "Free Riding on Gnutella Revisited: The Bell Tolls?" in *IEEE Distributed Systems Online*, vol. 6, no. 6, June 2005.
- [5] S. Jun and M. Ahamad, "Incentives in BitTorrent Induce Free Riding," in *P2P Econ*, August 2005.
- [6] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang, "Exploiting BitTorrent For Fun (But Not Profit)," in *IPTPS*, February 2006.
- [7] M. Sirivianos, J. H. Park, R. Chen, and X. Yang, "Free-riding in BitTorrent Networks with the Large View Exploit," in *IPTPS*, February 2006.
- [8] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, "Free Riding in BitTorrent is Cheap," in *HotNets*, November 2006.
- [9] M. Sirivianos, J. H. Park, X. Yang, and S. Jarecki, "Dandelion: Cooperative Content Distribution with Robust Incentives," in *USENIX*, June 2007.
- [10] B. Wilcox-O'Hearn, "Experiences Deploying a Large-Scale Emergent Network," in *IPTPS*, March 2002.
- [11] H. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin, "BAR Gossip," in *OSDI*, November 2006.
- [12] N. Asokan, M. Schunter, and M. Waidner, "Optimistic Protocols for Fair Exchange," in *ACM CCS*, April 1997.
- [13] J. Zhou and D. Gollmann, "An Efficient Non-repudiation Protocol," in *CSFW*, March 1996.
- [14] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," in *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, 2003, pp. 3–12.
- [15] "The eMule Project," www.emule-project.net.
- [16] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu, "Influences on Cooperation in Bittorrent Communities," in *P2P Econ*, August 2005.
- [17] "Kazaa Lite," en.wikipedia.org/wiki/Kazaa_Lite.
- [18] "NRPG RatioMaster: Fake Upload and Download Stats of a Torrent to Almost all Bittorrent Trackers," www.brothersoft.com/nrpg-ratiomaster-78031.html.
- [19] Q. Lian, Z. Zhang, M. Yang, B. Y. Zhao, Y. Dai, and X. Li, "An Empirical Study of Collusion Behavior in the Maze P2P File-Sharing System," in *ICDCS*, June 2007.
- [20] M. Piatek, T. Isdal, A. Venkataramani, and T. Anderson, "One Hop Reputations for File Sharing Workloads," in *NSDI*, April 2008.
- [21] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. A. Hamra, and L. Garces-Erice, "Dissecting BitTorrent: Five Months in a Torrents Lifetime," in *PAM*, April 2004.
- [22] B. Fan, D.-M. Chiu, and J. C. Lui, "The Delicate Tradeoffs in BitTorrent-like File Sharing Protocol Design," in *ICNP*, November 2006.
- [23] M. Bellare, R. Canetti, and H. Krawczyk, "Keying Hash Functions for Message Authentication," in *LNCS*, vol. 1109, 1996.
- [24] "The Transport Layer Security (TLS) Protocol, Version 1.1," <http://www.ietf.org/rfc/rfc4346.txt>.
- [25] J. Shneidman, C. Ng, D. C. Parkes, A. AuYoung, A. C. Snoeren, A. Vahdat, and B. N. Chun, "Why Markets Could (But Don't Currently) Solve Resource Allocation Problems in Systems," in *HotOS-X*, June 2005.
- [26] J. R. Douceur, "The Sybil Attack," in *IPTPS*, March 2002.
- [27] M. Sirivianos, X. Yang, and S. Jarecki, "Robust and Efficient Incentives for Cooperative Content Distribution," UCI, Tech. Rep., 2006, UCI-ICS TR 07-10. [Online]. Available: www.ics.uci.edu/~msirivia/publications/dandelion-ton-submission-tr.pdf
- [28] "Broadband promotions," <http://www.broadband-promotions.net/>.
- [29] "Verizon FiOS Internet Packages and Prices," <http://www22.verizon.com/content/consumerfios/>.
- [30] "Shop for bandwidth," www.shopforbandwidth.com/t1-lines-t3-lines-ds3-oc3-faste-gige-service.php.
- [31] D. Kotic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Maintaining High Bandwidth Under Dynamic Network Conditions," in *USENIX*, April 2005.
- [32] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do Incentives Build Robustness in BitTorrent?" in *NSDI*, April 2007.
- [33] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu, "Characterizing residential broadband networks," in *IMC*, October 2007.
- [34] K. Tamilmani, V. Pai, and A. Mohr, "SWIFT: A System with Incentives for Trading," in *P2P Econ*, August 2004.
- [35] I. Keidar, R. Melamed, and A. Orda, "EquiCast: Scalable Multicast with Selfish Users," in *PODC*, July 2006.
- [36] D. Levin, R. Sherwood, and B. Bhattacharjee, "Fair file swarming with FOX," in *Proc. 5th IPTPS*, April 2006.
- [37] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, "Karma: A Secure Economic Framework for P2P Resource Sharing," in *P2P Econ*, June 2003.
- [38] B. Yang and H. Garcia-Molina, "PPay: Micropayments for Peer-to-peer Systems," in *ACM CCS*, October 2003.
- [39] C. Aperijs, M. J. Freedman, and R. Johari, "Peer-Assisted Content Distribution with Prices," in *ACM CONEXT*, November 2008.

Michael Sirivianos is a Ph.D. candidate in Computer Science at the University of California, Irvine. His research interests include cooperative content distribution and human verifiable secure device pairing. He received a B.S. in Electrical and Computer Engineering from the National Technical University of Athens in 2002, and an M.S. in Computer Science from the University of California, San Diego in 2004.

Xiaowei Yang is an Assistant Professor of Computer Science at the University of California, Irvine. Her research interests include congestion control, quality of service, Internet routing architecture, and network security. She received a B.E. in Electronic Engineering from Tsinghua University in 1996, and a Ph.D. in Computer Science from MIT in 2004.

Stanislaw Jarecki is an Assistant Professor of Computer Science at the University of California, Irvine. His research interests are cryptography, security, and distributed algorithms. He received a B.S. in Computer Science from MIT in 1996, and a Ph.D. in computer science from MIT in 2001.

