CompactPSH: An Efficient Transitive TFT Incentive Scheme for Peer-to-Peer Networks

Bocek, T ; El-khatib, Y ; Victora Hecht, F ; Hausheer, D ; Stiller, B

DOI: https://doi.org/10.1109/LCN.2009.5355173

2009 IEEE 34th Conference on Local Computer Networks (LCN 2009)
Zürich, Switzerland; 20-23 October 2009

CompactPSH: An Efficient Transitive TFT Incentive Scheme for Peer-to-Peer Networks

Thomas Bocek, Fabio Victora Hecht
David Hausheer, Burkhard Stiller
Department of Informatics IFI, CSG,
University of Zurich, Switzerland
Email: bocek—hecht—hausheer—stiller@ifi.uzh.ch

Yehia El-khatib
Computing Department,
Lancaster University,
Lancaster LA1 4WA, United Kingdom
Email: yehia@comp.lancs.ac.uk

***Abstract*—Incentive schemes in Peer-to-Peer (P2P) networks are necessary to discourage free-riding. One example is the Tit-for-Tat (TFT) incentive scheme, a variant of which is used in BitTorrent to encourage peers to upload. TFT uses data from local observations making it suitable for systems with direct reciprocity. This paper presents CompactPSH, an incentive scheme that works with direct and indirect reciprocity. CompactPSH allows peers to establish indirect reciprocity by finding intermediate peers, thus enabling trade with more peers and capitalizing on more resources. CompactPSH finds transitive paths while keeping the overhead of additional messages low. In a P2P file-sharing scenario based on input data from a large BitTorrent tracker, CompactPSH was found to exploit more reciprocity than TFT which enabled more chunks to be downloaded. As a consequence, peers are allowed to be stricter to fight white-washing without compromising performance.**

I. Introduction

Peer-to-peer (P2P) systems have several advantages over centralized systems, including load balancing, robustness, scalability, and fault tolerance. However, open challenges still exist in P2P systems such as free-riders [1], malicious peers, Sybil attacks [6], self-interest [18], and other forms of attacks [13]. Free-riders, for instance, refuse to collaborate, which leads to overused resources and deterioration of service. Incentive mechanisms are used to address some of these challenges by promoting peers to follow certain rules such as acting cooperatively.

One incentive scheme is Tit-for-tat (TFT) which allows users to download only as much as they upload. This scheme is very popular and widely used. One example is a variant of TFT that is used in BitTorrent [4]. The TFT scheme keeps a private per-peer history of resource transactions that is solely based on local observation. However, the peers' view is limited to transactions of direct reciprocity and nothing else, restricting data exchange to a confined number of peers. A transitive TFT mechanism shares transaction information with other peers allowing indirect reciprocity to be detectable and exploitable. However, shared history approaches are generally prone to false reporting and collusion, or have scalability issues [8].

To overcome the above drawbacks, the CompactPSH incentive scheme is introduced. CompactPSH increases efficiency by employing a combination of private *and* shared history to integrate both direct and indirect reciprocity. CompactPSH uses Bloom filters [2] to discover intermediate peers through which interaction with more nodes (of indirect reciprocity) is made possible. CompactPSH thus facilitates trading resources between more peers while keeping messaging overhead lower than other approaches. As a consequence, the peers' credit limits can be decreased without lowering the performance. Thus, CompactPSH makes malicious threats such as white-washing more costly to carry out.

Our contribution is a theoretical analysis of CompactPSH and TFT, and an evaluation of the performance of CompactPSH and its effect on downloads by testing it against TFT and PSH_r [3]. In our experiments, a P2P file sharing application is simulated using input data from The Pirate Bay [9] and run on up to 968 peers over EMANICSLab [7].

The remainder of this paper is structured as follows. Section II discusses related work. Section III introduces the design of CompactPSH and Section IV provides the theoretical analysis. Section V provides the implementation details of CompactPSH and presents key results of CompactPSH in comparison with TFT and PSH_r. Finally, Section VI draws conclusions and outlines future work.

II. Related Work

Incentive schemes can be classified into trust-based and trade-based incentive schemes [14]. Since CompactPSH is a trade-based scheme, this section focuses on this class.

A. Literature Review

BarterCast [11] is a recently proposed reputation mechanism that aims to enforce a long-term balanced sharing-ratio for all peers in a BitTorrent file sharing network. It addresses asymmetric interest by spreading up- and download statistics among neighbor peers. The MaxFlow algorithm is used to limit the effect of malicious peers. The approach is similar to that of PSH, PSH_r [3] and CompactPSH as these mechanisms try to find a transitive path and exchange peer information. While BarterCast is integrated in BitTorrent and uses an epidemic protocol to spread history information, PSH, PSH_r and CompactPSH uses its own file sharing application and attaches history information to existing messages.

Give-to-Get [12] (GtG) focuses on reducing free-riding in P2P Video-on-Demand systems. Using GtG, a source peer is

encouraged to upload to target peers that upload more to third party peers. This is achieved by reporting upload statistics from the third party peers to the source peer. However, as those reports come from third party peers, which may collude with target peers, GtG is prone to false upload statistic reports.

Roger et al. [17] use game-theoretical approaches to compare between TFT, Stochastic TFT (STFT), BitTorrent, reactive strategies, and expected utility strategy (EXU). In their setup, they demonstrate that Stochastic TFT and EXU are evolutionarily stable, which means that those strategies remain stable if other strategies are applied. Furthermore, Roger et al. also show that peers have a higher payoff when using STFT or EXU rather than the BitTorrent approach.

Feldman et al. [8] propose to use the MaxFlow algorithm for a robust incentive mechanism. The authors suggest modifying the MaxFlow algorithm to search and evaluate paths in constant time. As MaxFlow cannot be computed in constant time, only a subset of paths are found due to this complexity reduction.

Ngan et al. [21] present an auditing architecture to enforce fair sharing of storage resources, which is robust against collusion. Audit information and usage records are publicly available and any peer can audit any other peer. The authors show through simulation that the auditing overhead is small, that their scheme scales well in large networks, and that peers have an incentive to provide correct data.

B. Discussion

A key difference between CompactPSH and BarterCast is that BarterCast focuses on integration with BitTorrent. In the current BarterCast implementation, the transferred history information are top-10 peers only. This information is spread periodically to neighbor peers using an epidemic protocol. This approach suffers if history information gets outdated [3]. CompactPSH uses a Bloom filter-based algorithm to exchange peer information, which generates smaller overhead and allows the exchange of much more history information. Moreover, history information encoded in Bloom filters is attached to reply messages to keep the history information up to date while avoiding the need to create new connections.

The GtG incentive scheme aims to optimize resource distribution by encouraging peers to select destination peers that offer faster data exchange. Thus, this incentive scheme is not based on reciprocity since a source peer is interested in providing resources without consuming. Rogers et al. present and compare several mechanisms. In their paper, the TFT mechanism is termed STFT, and the authors show that this is one of the better mechanisms with respect to evolutionary stability and payoff. The path-finding algorithm presented by Feldman et al. requires contacting many peers for the MaxFlow algorithm to find a path. CompactPSH first finds paths, then applies MaxFlow, which requires fewer peer contacts. The work presented by Ngan et al. uses auditing to verify resource information, while CompactPSH tries to find a transitive resource exchange path.

TABLE I
RELATED WORK COMPARISON

Mechanism	*Reciprocity*	*Indirect Reciprocity*	*One Intermediate Hop*	*Collusion Resistant*	*Efficient Hop Search*
BarterCast	yes	yes	yes	yes	no
GtG	no	no	yes	no	n/a
(S)TFT	yes	no	no	yes	n/a
Feldman et al.	yes	yes	no	yes	no
Ngan et al.	no	no	no	yes	n/a
PSH	yes	yes	no	yes	no
PSH_r	yes	yes	yes	yes	no
CompactPSH	yes	yes	yes	yes	yes

The key difference between CompactPSH and PSH [3] is that CompactPSH focuses on transitive paths with one intermediate peer, while PSH supports multiple intermediate peers. Furthermore, in PSH [3], a peer sends a subset of candidates to any peers as an attachment to all messages, while in CompactPSH, the target peer sends such an attachment only if a resource request has failed.

With PSH_r, target peers send a subset of candidates while CompactPSH encodes all candidates. CompactPSH is able to find more intermediate peers since more peers can be encoded in the Bloom filter than in the PSH_r subset with the same size. On the down side, using Bloom filters might result in finding false positive intermediate peers, i.e. ones that are not on a transitive path. For false positive intermediate peers, two additional messages are exchanged.

Table I presents a comparison of the aforementioned related work, highlighting whether the different schemes deal with the following aspects: reciprocity, indirect reciprocity, search for one intermediate hop, collusion resistance, and efficient hop search. A hop search is not applicable (n/a) for GtG, (S)TFT, and Ngan et al. as these do not support indirect reciprocity.

III. DESIGN

Hereafter, the following terminology is used. Peers that request resources are termed *source peers* (S), while target peers of such requests are termed *target peers* (T). Peers that are neither source nor target peers and are in a transitive path between source and target peers are termed *intermediate peers* (I). A *transitive path* includes three or more peers with indirect reciprocity. Potential intermediate peers are termed *source_candidates* or *target_candidates*. A *source_candidate* has previously received resources from the source peer, while a *target_candidate* has previously provided resources to the target peer. *Credit limit* is a threshold of data units that each peer can download without uploading. A *check* is a signed message with a credit value, and source and target peer IDs.

A. CompactPSH Design

The CompactPSH algorithm has four different phases. In every phase, two peers interact with one another using *request*/*reply* messages. The following list presents an overview of these phases.

- Phase 1: Request and provide resources while collecting and aggregating private history information.

- Phase 2: Search for a one-hop transitive path using private history information.
- Phase 3: If intermediate peers are found, balance history accounts of source and target peer considering MaxFlow [5] on intermediate peers and issue checks.
- Phase 4: Re-request resources using checks.

In the first phase, public keys are exchanged on first contact with unknown peers. While peers consume resources from one another, CompactPSH builds private history the same way TFT does. Hence, private histories are updated with peers in direct reciprocity. If the credit limit of a peer requesting resources is exceeded, the target peer verifies this from its private history and subsequently denies the request. This transaction would fail under TFT.

CompactPSH, however, initiates a second phase in which it searches for a one-hop transitive path between the source and target peers. If the credit limit is reached, the target peer sends a *request denied* message to the source peer along with a Bloom filter [2] encoding the IDs of all *target_candidates*. The source peer then tests its *source_candidates* against the received Bloom filter by iterating over its *source_candidates* list. Matching candidates are chosen as intermediate peers. The complexity of this phase is dictated by the size of the candidate list. Using Bloom filters, more potential intermediate peer IDs can be encoded into an array as compared to sending the IDs in an array with same length.

The third phase commences once the search for an intermediate peer is successful. In this phase, an intermediate peer is requested to balance its history accounts for the source and target peers, and to issue a check and send it back to the source peer. The intermediate peer balances these history accounts by decreasing the account of the source peer and increasing the account of the target peer. Public keys exchanged during Phase 1 are used by the intermediate peer to sign the check in order to prevent forgery.

The fourth phase starts after the source peer receives a signed check from the intermediate peer. The source peer attaches the check to a request message. Upon receiving this, the target peer verifies the signature and applies the check before it processes the request. Handling the request is as described in the first phase and, depending on the target peer's private history, the request will be either granted or denied.

An example of these four phases is shown in Figure 1. In phase 1, peer I receives one resource from peer S while peer T receives one resource from peer I. In phase 2, peer I is requested to balance the history for peers S and T. In phase 3, peer S requests and receives a check from peer I. In this example, the history shows accounts after peer S has applied the check. In phase 4, peer S re-requests resources from peer T. The private histories show accounts after peer T has received the check but before peer T replies.

IV. Theoretical Analysis

This section offers a theoretical analysis that compares the exploitation probability of indirect and direct reciprocity (as in PSH and CompactPSH) to direct reciprocity (as in TFT).

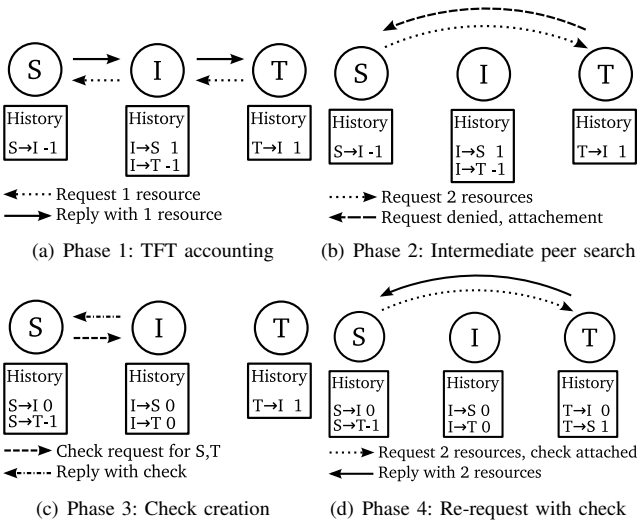

Fig. 1. Example CompactPSH history with a source peer (S), intermediate peer (I), and target peer (T)

Let A, B, C, and D be four peers, where A provided a resource to B (B consumed from A), B provided to C (C consumed from B), and C provided to D (D consumed from C). p_{xy} defines the probability that a peer X can provide resources to peer Y, or a peer Y can consume resources from peer X. For example, $p_{ab} = 0.9$ indicates that peer A can provide resources to peer B with 90% probability. It is assumed that paths and intermediate peers are known. In an ideal situation where $p = 1$, CompactPSH would not be necessary as direct reciprocity can always be exploited. However, in reality, $p < 1$ because peers may not have the requested resources. For instance, in a file sharing application a peer is likely to have no chunks required by a another peer. Figure 2 shows peers A, B, C, and D and their probabilities. The probabilities that D consumes from C, C consumes from B, and B consumes from A are all 100% ($p_{ab} = p_{bc} = p_{cd} = 1$). With transitivity, $p_{ac} = p_{ad} = p_{bd} = 1$.

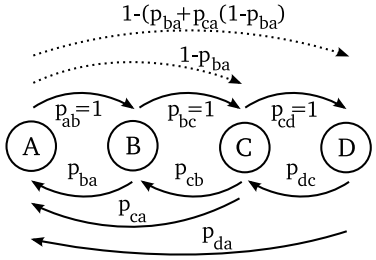

Fig. 2. Probabilities for peer A, B, C, and D. Solid line are probabilities for consumption / provision, dashed lines are requests for consumption / provision

Exploitable reciprocity r defines how much reciprocity can be exploited, *e.g.*, exploitable reciprocity between A and B is defined as $p_{ba} = rp_{ab}$ and for the values $r = 0.5$ and $p_{ab} = 1$, B can provide resources to A with a 50% probability, while A can provide resources to B with 100% probability.

With indirect reciprocity, peer A queries C with probability

$1 - p_{ba}$, as A failed to consume from B. C can provide resources to A with probability p_{ca}, where $p_{ca} = rp_{ac}$. Thus, the probability for A to successfully consume from B or C with direct and one-hop reciprocity is $p_{ba} + ((1 - p_{ba})p_{ca})$, where p_{ba} is the probability to exploit direct reciprocity and $(1 - p_{ba})p_{ca}$ exploits a one-hop indirect reciprocity. Exploiting direct and one-hop reciprocity fails with probability $1-(p_{ba}+((1-p_{ba})p_{ca}))$, which is the probability to contact D. Consequently, the probability to exploit two-hop indirect reciprocity is $(1 - (p_{ba} + ((1 - p_{ba})p_{ca})))p_{da}$. Thus, the equation for exploiting reciprocity with x hops and exploitable reciprocity r is shown in Equation 1.

$$f_x(r) = \begin{cases} r & \text{if } x = 0, \\ r(1 - f_{x-1}) + f_{x-1} & \text{if } x > 0. \end{cases} \quad (1)$$

Figure 3 shows a comparison of exploiting direct reciprocity f_0 as in TFT, exploiting direct and one-hop indirect with reciprocity f_1 as in CompactPSH and PSH_r, and exploiting direct and one and two-hop indirect reciprocity f_2 as in PSH with $r = [0, 1]$. The f_1 improvement over f_0 is defined as $1 - \frac{f_1}{f_0}$. The f_0 and f_1 comparison graph shows that for $r = 10\%$, $f_1(0.1) = 19\%$, which is 90% better than $f_0(0.1) = 10\%$. For a high r, *e.g.*, $r = 90\%$, $f_1(0.9) = 99\%$, which is 10% better than $f_0(0.9) = 0.9\%$. Thus, a mechanism that exploits indirect reciprocity works better for small r. The total exploitation for f_2 with up to two hops is higher than for f_1. However, the marginal total exploitation for f_2 is smaller than for f_1 and path finding with one hop involves less peers than with two hops. Furthermore, longer paths tend to be less stable as history information may be outdated.

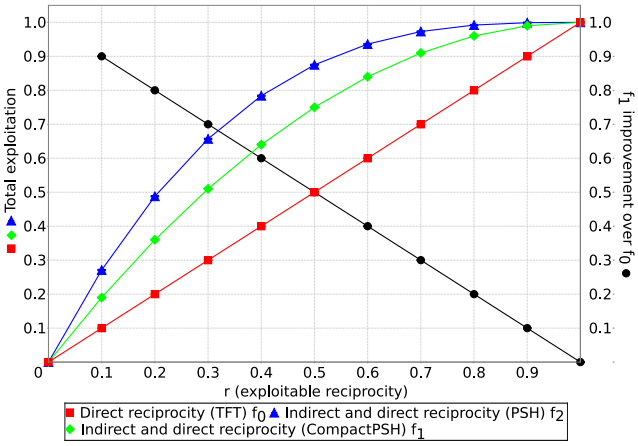

Fig. 3. Indirect and direct reciprocity comparison

V. Implementation and Evaluation

Experiments have been run to compare CompactPSH against PSH_r and TFT. The outcome of these experiments gives a basis to evaluate scalability, adaptability, robustness, and overhead of each of these incentive schemes. To run these experiments, a file sharing application has been developed to implement the three incentive schemes.

In each experiment, various combinations of the following operating conditions were used: number of peers, number of free-riders, chunk size, credit limit, goodwill, and file multiplicator. The number of peers n refers to those peers that participate in the network and publish all chunk availabilities, while the number of free-riders f describes those peers that refrain from publishing their chunk availabilities. Each file is divided into chunks of uniform size, *i.e.*, the chunk size c. The implementation aggregates transaction information on a per-peer basis. In the experiments, the maximum number of peers is 968 peers. Goodwill g is the probability of a peer receiving a chunk it requested even after the point where the credit limit l has been reached. For example, a goodwill value of 10% gives a peer that has exhausted its credit a 10% chance of downloading another chunk. The file multiplicator m determines the number of files in the network. A summary of these parameters and their default values is found in Table II.

TABLE II
Experiment Parameters

Symbol	*Parameter*	*Default Value*
n	Number of peers	506
c	Chunk size	2,000 Bytes
m	File multiplicator	3
l	Credit limit	8,000 Bytes
g	Goodwill	10%
f	Free-riders	0%

A. Implementation

The implemented file sharing application supports publishing and downloading files. Publishing a file requires first to search for a tracker. A tracker is responsible for keeping track of all peers that have at least one chunk of a file with the file content hash closest to the tracker ID. If a tracker exists, the address of the publishing peer is added to the tracker. A tracker is created if it does not exist.

Downloading a file starts with searching for its key. The application supports a simple search for keys using the file name as a search query. Then, the key of the file is used to find trackers for the file. If a tracker is found, it is queried for the addresses of a random set of peers that have at least one chunk. These peers are then queried for chunk availability. Comparing remote and local chunk availabilities shows which chunks can be downloaded from the remote peers. Accordingly, remote peers are requested to send a random chunk of the desired ones. If there is sufficient credit, download commences. If not, TFT, PSH_r and CompactPSH are used to find means to trade credits. TFT only explores direct reciprocity while PSH_r and CompactPSH explore both direct and indirect reciprocity.

Thus, each successfully downloaded chunk is due to initial credit, direct/indirect reciprocity, goodwill, or being a *potential trader*. A potential trader is a peer with direct or indirect reciprocity that has chunks to provide. If a potential trader requests a chunk, the request is granted because the credit may be traded in the future.

Depletion of a peer's credit signals the end of data provision to it. This is preferred to other practices such as lowering QoS, *e.g.*, lowering the bandwidth, as our main interest is to analyze indirect reciprocity.

In the current implementation, no peer reports wrong chunk availability to exploit being a potential trader. In such a situation, peers could dynamically adjust a limit for potential trader to limit the negative effects of reporting wrong chunk availability.

The implementation of Bloom filters is as follows. If a peer requests a list of candidates, a 10,399 bit long Bloom filter is created to encode the peer IDs of all candidates. This length has been chosen to compare CompactPSH to PSH_r, which transfers 1,300 bytes of candidates (10,399/8 $\approx$ 1,300). With 1,000 peers, the expected probability of a false positive is 0.68%. This probability decreases as the number of peers decreases. To maintain a low false positive probability with more peers, the Bloom filter bit length has to be increased. In case of a false positive, one *request* and one *request denied* message are exchanged.

B. Testbed Infrastructure

To provide an evaluation testbed with a high degree of realistic characteristics, EMANICSLab [7] has been chosen to run the experiments. EMANICSLab is a research network established among European research partners. It consists of 20 hosts at 10 different partner sites. EMANICSLab uses MyPLC [7], a technology developed for PlanetLab [16], to manage virtual servers on host machines. During our experiments, 11 out of 14 hosts were online.

It is worth noting that EMANICSLab is not a dedicated infrastructure and it is made up of heterogeneous machines with different hardware configurations, i.e., CPU models and RAM size. Hence, the delivered computational performance may change due to contention with other experiments.

C. Input Data

For a more realistic experimental P2P environment, data from a real P2P file sharing network was used because parameters, such as file size and popularity, have a significant impact on the experiments. Our input data was gathered from The Pirate Bay [20], one of the most popular BitTorrent trackers with (at the time of writing) more than 1,900,000 files shared between about 18 million peers [9]. The site was queried and torrent metadata were indexed, specifically file name, file size, number of seeders, and number of leechers. A subset of 337,996 torrents was indexed, representing approximately 26% of the total number of torrents reported by the tracker at the time of retrieval. The average file popularity, shown in Figure 4, was used to model the experiment. Popularity refers to number of seeders plus leechers. This data is available from [9].

D. Experimental Settings

In the experiments, 29 files were published using the file size distribution from the input data (cf. Section V-C). The file multiplicator m is used to change the number of files in different experiments. File sizes have been scaled down by a factor of 1,000 to keep the experiment time under three days. The user behavior is modeled as follows. Each user requested 3 files according to the file popularity distribution from the input data, and stopped downloading after 30 minutes.

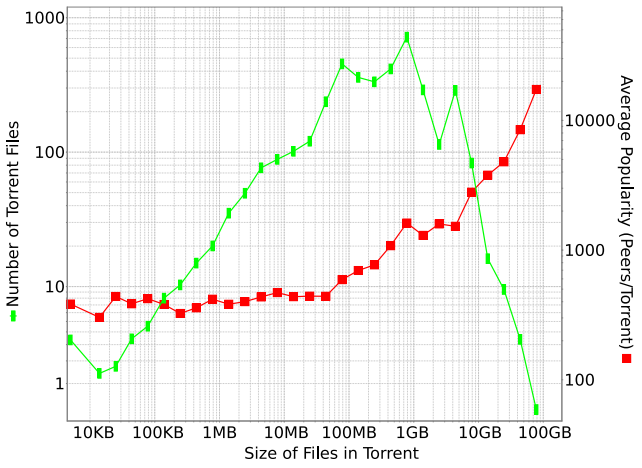

Fig. 4. Distribution, popularity, and file sizes from The Pirate Bay [9]

In the following experiments, the parameters $n : [462, 968]$, $c = 2000$, $m : [1, 8]$, $l : [2000, 12000]$, $g : [0, 55\%]$, $f : [0, 90\%]$ are used. In each experiment, one parameter is changed while the remaining parameters are kept constant within their respective ranges (cf. Table II). This procedure helps in investigating the effect of one variant at a time.

E. Evaluation Results

All results are plotted along with the standard deviation of each measurement point, which covers the deviation value for 5 different test runs. The message size per downloaded chunk contains chunk request and reply, chunk availability request and reply, and check messages, including header information. The default initial credit limit l is set to 8,000 Bytes thus each peer can download 4 chunks before its credit runs out.

The first experiment investigates how the three incentive schemes react to free-riders. Free-riders are peers that download but refuse to upload. The following parameter was changed: $f : [0, 80\%]$. Figure 5 shows that the number of downloaded chunks decreases as the number of free-riders increase *i.e.* fewer peers are involved in trading. An increasing number of free-riders is a burden on the P2P system coming at the cost of an increasing amount of unsuccessful chunk requests. This hinders the performance gained by all three incentive schemes, as portrayed in Figure 5, yet CompactPSH continues to offer higher download rates than PSH_r, which in turn performs better than TFT. However, beyond 20% free-riding peers no clear winner could be determined. Figure 6 outlines the increasing overhead as the number of free-riders increase. This is calculated as the total size of messages divided by the downloaded chunks, which is the number of

messages needed to download one chunk. From the figure it is observed that CompactPSH has less overhead then PSH_r for free-riders between 0% and 30%, but the two schemes have similar overhead afterwards. This is due to the same size for candidates which did not contribute to additional chunk downloads. Furthermore, the overhead for TFT is always smaller than for PSH_r and CompactPSH because no checks and candidate peers are exchanged.

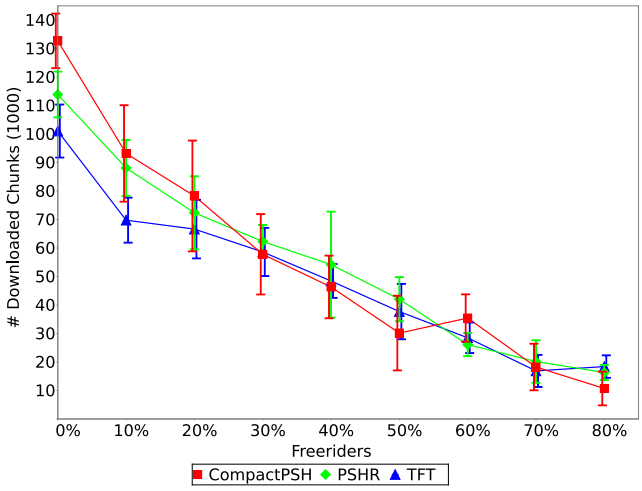

Fig. 5. Number of downloaded chunks for different number of free-riders – f:[0,80%]

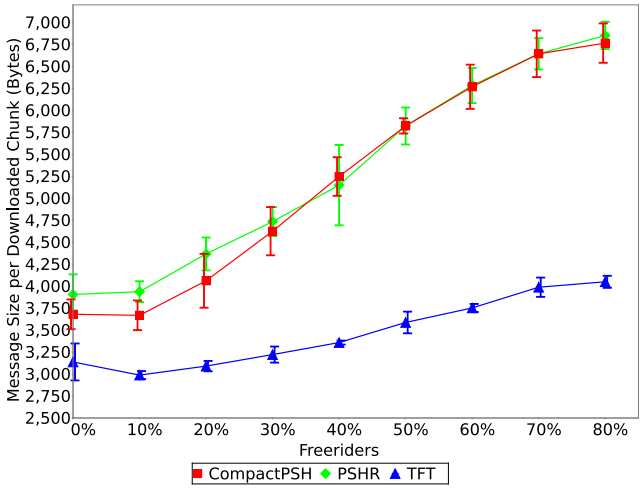

Fig. 6. Overhead for different number of free-riders – f:[0,80%]

As mentioned earlier, a high goodwill value means that peers are more tolerant which results in a larger number of downloaded chunks. With lower goodwill, less peers are involved in finding transitive paths resulting in a lower chance of finding an intermediate peer. With a higher goodwill, peers can download more chunks without relying on transitive paths. Thus, it is important to find a suitable goodwill value. To examine the effects of goodwill on the different incentive mechanisms, the following parameter was changed: $g : [0, 55\%]$. Figure 7 illustrates the outcome of this experiment. It is observed that using CompactPSH results in the same number of downloaded chunks as that achieved using TFT with a higher goodwill. In other words, CompactPSH allows peers to be stricter without negatively affecting the download rate. For CompactPSH, goodwill around the 20%-30% range has the highest benefit compared to TFT.

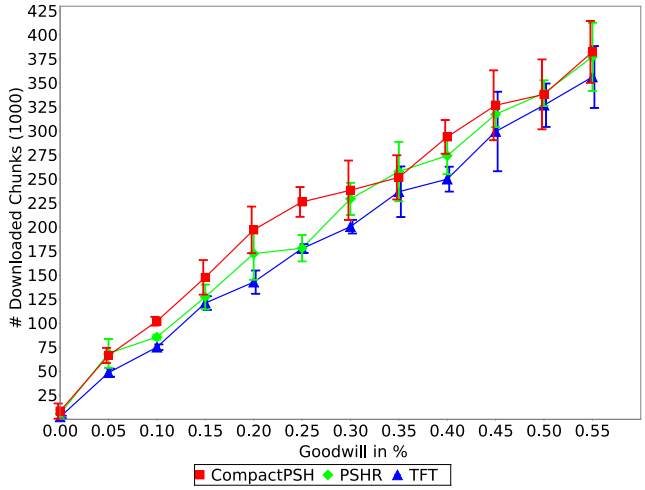

Fig. 7. Number of downloaded chunks for different goodwill values – g:[0,55%]

To test the scalability of the three incentive schemes, performance and overhead are measured as the number of peers n increases from 462 (11 machines, 42 peers per machine) to 968 (88 peers per machine), $n : [462, 968]$. Figure 8 shows that CompactPSH always downloads more chunks with increasing number of peers than both PSH_r and TFT. On average, CompactPSH downloads 14.7% more chunks than PSH_r and 24.2% more chunks than TFT. Figure 9 depicts how CompactPSH requires less messaging overhead per downloaded chunk than PSH_r due to the use of Bloom filters, while TFT needs less messages than CompactPSH and PSH_r. This overhead is explained due to searching for and finding intermediate peers. While more intermediate peers can be found with CompactPSH than with PSH_r, more chunks can be downloaded, thus the overhead decreases. While TFT uses less than 2,900 bytes, CompactPSH uses up to 3,400 bytes per downloaded chunk, a 17.2% larger message size. Although the overhead seems relatively high, it is important to mention that the overhead decreases with a larger chunk size. These experiments use a chunk size of 2,000 bytes and the size of both the PSH_r candidate list and the CompactPSH Bloom filter is 1,300 bytes.

The next experiment is to study how the incentive schemes adapt to an increase in the credit limit, which clearly allows more chunks to be downloaded. The following parameter was changed: $l : [2000, 12000]$. Figure 10 shows that CompactPSH downloaded more chunks than both PSH_r and TFT. It can be observed that CompactPSH can download the same amount of chunks with a lower credit limit than both TFT and PSH_r.

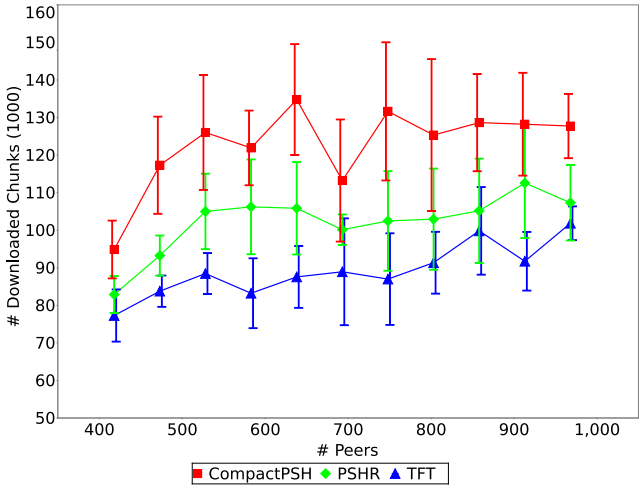

Fig. 8. Number of downloaded chunks for different number of peers – n:[462,968]

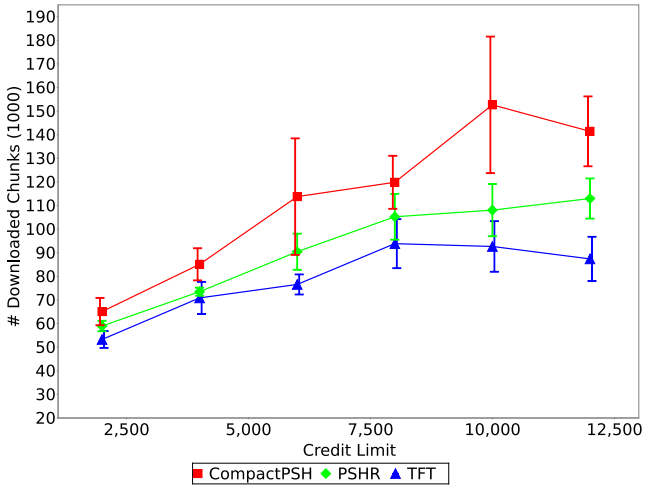

Fig. 10. Number of downloaded chunks for different credit limits – l:[2000,12000]

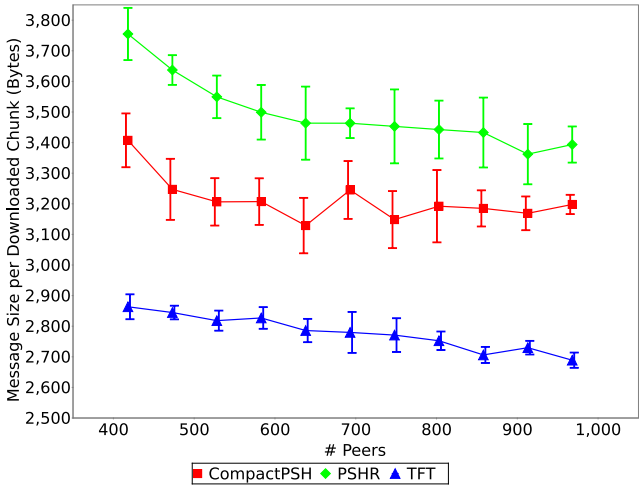

Fig. 9. Overhead for different number of peers – n:[462,968]

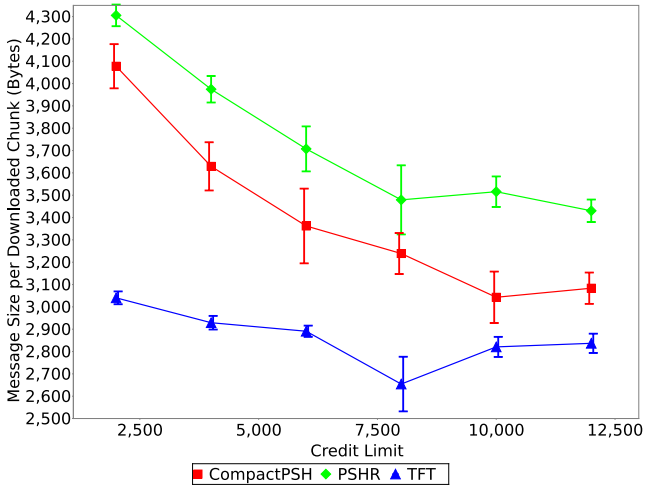

Fig. 11. Overhead for different credit limits – l:[2000,12000]

The number of chunks downloaded with CompactPSH seem to continuously increase as the credit limit increases, apart from the result at 10,000 which seems to be a slightly inflated value as indicated by the error bars. On the other hand, the number of chunks downloaded by TFT seems to peak at 7,500 and drop afterwards. What accounts for this slight decline in performance is that a peer can download smaller files from only one peer. Thus, less reciprocities are established from smaller files. This also applies for PSH_r and CompactPSH. In Figure 11, the number of messages per downloaded chunk decreases as less chunk failures happen. The overhead for TFT is smaller than CompactPSH, while the overhead for CompactPSH is smaller than PSH_r as observed previously in Figure 9.

Finally, the effects of varying the number of files are studied. The following parameter was changed: $m : [1, 9]$. Figure 12 illustrates the number of downloaded chunks as the file multiplicator changes. More files decrease the number of downloaded chunks because fewer peers offer the same file. Thus, fewer peers are involved in the same download and fewer peers can trade. Figure 12 shows that CompactPSH downloaded more chunks than both PSH_r and TFT as observed in the previous Figures.

VI. Conclusions and Future Work

CompactPSH is an efficient incentive scheme that makes use of private and shared history information to incorporate both direct and indirect reciprocity. CompactPSH reduces messaging overhead by using Bloom filters. CompactPSH uses MaxFlow [5] to limit resource over-usage as shown in [8], [10]. This results in a broadened group of peers that a peer can interact with and reduces the risk of malicious attacks such as white-washing, free-riding and collusion.

CompactPSH was evaluated against PSH_r and TFT in a P2P file sharing scenario. The experiments show that Com-

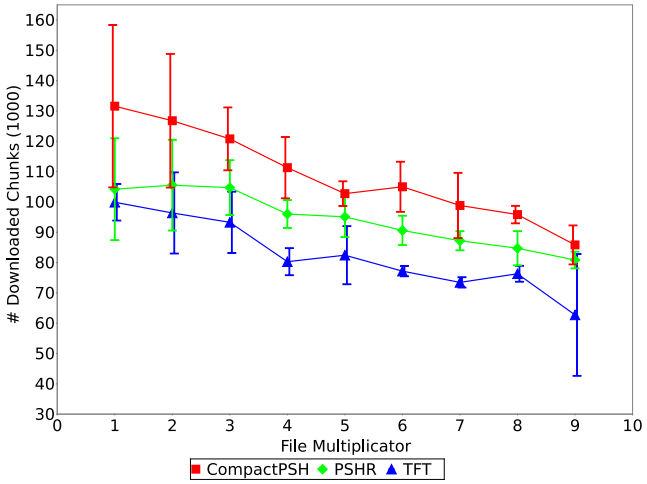

Fig. 12. Number of downloaded chunks for different number of files – m:[1,9]

pactPSH achieves the same number of downloaded chunks with lower goodwill and with lower initial credit limit. Thus, with lower credit limit and goodwill values, CompactPSH makes threats such as white-washing more expensive, since new identities need to be created more often. With an increasing number of free-riders, CompactPSH performs better up to 20% free-riding peers and performs just as well as PSH_r and TFT beyond that. Malicious and colluding peers might attempt to report wrong history data. However, since CompactPSH applies MaxFlow [5], the effects of such behavior is limited.

Future work will investigate locality issues in the file sharing application. As described in Section V-A, peers are randomly chosen, hence locality is not considered. The tracker could perform an initial peer selection based on locality, as suggested in [15] and [19], to reduce transit costs for the ISP and to improve application performance. Additionally, for the TFT, PSH_r and CompactPSH incentive schemes, a Bloom filter describing the set of peers that are of interest to a certain peer could be sent to the tracker. Thus, it is feasible to modify the peer selection scheme to be based on the interest of the requesting peer and ISPs. Another interesting discussion is the lack of incentives to provide additional resources once peers get the requested resources. An incentive scheme to get resources faster is a benefit for the providing peer, but is a disadvantage for other consuming peers as less peers provide resources and the availability decreases. Another area of potential improvement is dealing with inconsistent histories. During CompactPSH phases, histories may become inconsistent. An example of such inconsistent histories is shown in Figure 1(c), where the history of peer T reports an upload to peer I and peer I reports no upload. Furthermore, future work includes simulations and experiments with more peers, more simulation and experiment runs and developing the incentive mechanism further for delay sensitive applications such as P2P video streaming.

ACKNOWLEDGEMENTS

This work has been performed partially in the framework of the EU IST Project EC-GIN (FP6-2006-IST-045256) as well as the EU IST NoE EMANICS (FP6-2004-IST-026854).

REFERENCES

[1] E. Adar and B. A. Huberman. Free Riding on Gnutella. *First Monday, Internet Journal*, 5(10), October 2000.
[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
[3] T. Bocek, W. Kun, F. V. Hecht, D. Hausheer, and B. Stiller. PSH: A Private and Shared History-based Incentive Mechanism. In *2nd International Conference on Autonomous Infrastructure, Management and Security Resilient Networks and Services (AIMS 2008)*, Bremen, Germany, July 2008.
[4] B. Cohen. Incentives Build Robustness in BitTorrent. In *1st Workshop on Economics of Peer-to-Peer Systems (P2PECON)*, Berkeley, CA, USA, June 2003.
[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.
[6] J. R. Douceur. The sybil attack. In *First International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, pages 251–260, London, UK, 2002. Springer-Verlag.
[7] EMANICSLab. http://emanicslab.org, 2009.
[8] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust Incentive Techniques for Peer-to-Peer Networks. In *5th ACM conference on Electronic commerce (EC '04)*, pages 102–111, New York, NY, USA, 2004. ACM Press.
[9] F. Hecht, T. Bocek, and D. Hausheer. The Pirate Bay 2008-12 Dataset. http://www.csg.uzh.ch/publications/data/piratebay/, December 2008.
[10] R. Landa, D. Griffin, R. G. Clegg, E. Mykoniati, and M. Rio. A Sybilproof Indirect Reciprocity Mechanism for Peer-to-Peer Networks. In *28th IEEE International Conference on Computer Communications (INFOCOM 2009)*, Rio de Janeiro, Brasil, April 2009.
[11] M. Meulpolder, J. Pouwelse, D. Epema, , and H. Sips. BarterCast: Fully Distributed Sharing-Ratio Enforcement in BitTorrent. Technical report, Delft University of Technology, 2008.
[12] J. J. D. Mol, J. A. Pouwelse, M. Meulpolder, D. H. J. Epema, and H. J. Sips. Give-to-Get: free-riding resilient video-on-demand in P2P systems. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 6818, Jan. 2008.
[13] S. J. Nielson, S. Crosby, and D. S. Wallach. A Taxonomy of Rational Attacks. In *4th Annual International Workshop on Peer-To-Peer Systems (IPTPS 2005)*, Ithaca, New York, USA, February 2005.
[14] P. Obreiter and J. Nimis. A Taxonomy of Incentive Patterns - The Design Space of Incentives for Cooperation. In *Second International Workshop on Agents and Peer-to-Peer Computing (AP2PC'03)*, Melbourne, Australia, July 2003. Springer LNCS 2872.
[15] P4P - Enable ISP & P2P to Work Together. http://www.openp4p.net, 2009.
[16] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir. Experiences building planetlab. In *7th USENIX Symposium on Operating System Designand Implementation (OSDI '06)*, November 2006.
[17] M. Rogers and S. Bhatti. Cooperation under Scarcity: The Sharer's Dilemma. In *2nd International Conference on Autonomous Infrastructure, Management and Security Resilient Networks and Services (AIMS 2008)*, Bremen, Germany, July 2008.
[18] J. Shneidman and D. C. Parkes. Rationality and Self-Interest in Peer to Peer Networks. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS 2003)*, Berkeley, CA, USA, February 2003.
[19] SmoothIT - Simple Economic Management Approaches of Overlay Traffic in Heterogeneous Internet Topologies. http://www.smoothit.org, 2009.
[20] The Pirate Bay. http://thepiratebay.org/, 2009.
[21] D. S. W. Tsuen-Wan Ngan and P. Druschel. Enforcing Fair Sharing of Peer-to-Peer Resources. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS 2003)*, Berkeley, CA, USA, February 2003.