LiFTinG: Lightweight Freerider-Tracking in Gossip[★]

Rachid Guerraoui[1], Kévin Huguenin[2], Anne-Marie Kermarrec[3],
Maxime Monod[1], and Swagatika Prusty[4]

[1] Ecole Polytechnique Fédérale de Lausanne
[2] IRISA / Université of Rennes 1
[3] INRIA Rennes-Bretagne Atlantique
[4] IIT Guwahati

Abstract. This paper presents LiFTinG, the first protocol to detect freeriders, including colluding ones, in gossip-based content dissemination systems with asymmetric data exchanges. LiFTinG relies on nodes tracking abnormal behaviors by cross-checking the history of their previous interactions, and exploits the fact that nodes pick neighbors at random to prevent colluding nodes from covering up each others' bad actions.

We present a methodology to set the parameters of LiFTinG based on a theoretical analysis. In addition to simulations, we report on the deployment of LiFTinG on PlanetLab. In a 300-node system, where a stream of 674 kbps is broadcast, LiFTinG incurs a maximum overhead of only 8% while providing good results: for instance, with 10% of freeriders decreasing their contribution by 30%, LiFTinG detects 86% of the freeriders after only 30 seconds and wrongfully expels only a few honest nodes.

1 Introduction

Gossip protocols have recently been successfully applied to decentralize large-scale high-bandwidth content dissemination [5,7]. Such systems are *asymmetric*: nodes propose packet identifiers to a dynamically changing random subset of other nodes. These, in turn, request packets of interest, which are subsequently pushed by the proposer. In such a three-phase protocol, gossip is used to disseminate content location whereas the content itself is explicitly requested and served. These protocols are commonly used for high-bandwidth content dissemination with gossip, e.g., [5,7,8,21] (a similar scheme is also present in mesh-based systems, e.g., [20, 25, 26]).

The efficiency of such protocols highly relies on the willingness of participants to collaborate, i.e., to devote a fraction of their resources, namely their upload bandwidth, to the system. Yet, some of these participants might be tempted to *freeride* [19], i.e., not contribute their fair share of work, especially if they can still

benefit from the system. Freeriding is common in large-scale systems deployed in the public domain [1] and may significantly degrade the overall performance in bandwidth-demanding applications. In addition, freeriders may collude, i.e., collaborate to decrease their individual contribution and the contribution of the coalition and cover up each other's misbehaviors to circumvent detection mechanisms.

By using the Tit-for-Tat (TfT) incentives (inspired from file-sharing systems [4]), content dissemination solutions (e.g., [21]) force nodes to contribute as much as they benefit by means of balanced *symmetric* exchanges. As we review in related work (Section 7), those systems do not perform as well as *asymmetric* systems in terms of efficiency and scalability.

In practice, many proposals (e.g., [5, 20, 25, 26]) consider instead asymmetric exchanges where nodes are supposed to altruistically serve content to other nodes, i.e., without asking anything in return, where the benefit of a node is not directly correlated to its contribution but rather to the global *health* of the system. The correlation between the benefit and the contribution is not immediate. However, such correlation can be artificially established, in a coercive way, by means of verification mechanisms ensuring that nodes which do not contribute their fair share do no longer benefit from the system. Freeriders are then defined as nodes that decrease their contribution as much as possible while keeping the probability of being expelled low.

We consider a generic three-phase gossip protocol where data is disseminated following an asymmetric push scheme. In this context, we propose LiFTinG, a lightweight mechanism to track freeriders. To the best of our knowledge, LiFTinG is the first protocol to secure asymmetric gossip protocols against possibly colluding freeriders.

At the core of LiFTinG lies a set of deterministic and statistical distributed verification procedures based on *accountability* (i.e., each node maintains a digest of its past interactions). Deterministic procedures check that the content received by a node is further propagated following the protocol (i.e., to the right number of nodes within a short delay) by cross-checking nodes' logs. Statistical procedures check that the interactions of a node are evenly distributed in the system using statistical techniques. Interestingly enough, the high dynamic and strong randomness of gossip protocols, that may be considered as a barrier at first glance, happens to help tracking freeriders. Effectively, LiFTinG exploits the very fact that nodes pick neighbors at random to circumvent collusion: since a node interacts with a large subset of the nodes, chosen at random, this drastically limits its opportunity to freeride without being detected, as it prevents it from deterministically choosing colluding partners that would cover it up.

LiFTinG is lightweight as it does not rely on heavyweight cryptography and incurs only a very low overhead in terms of bandwidth. In addition, LiFTinG is fully decentralized as nodes are in charge of verifying each others' actions. Finally, LiFTinG provides a good probability of detecting freeriders while keeping the probability of false positive, i.e., inaccurately classifying a correct node as a freerider, low.

We give analytical results backed up with simulations, providing means to set the parameters of LiFTinG in a real environment. Additionally, we deployed LiFTinG over PlanetLab, where a stream of 674 kbps is broadcast to 300 PlanetLab nodes having their upload bandwidth capped to 1000 kbps for the sake of realism. In the presence of freeriders, the health of the system (i.e., the proportion of nodes able to receive the stream in function of the stream lag) degrades significantly compared to a system where all nodes follow the protocol. Figure 1 shows a clear drop between the plain line (no freeriders) and the dashed line (25% of freeriders). With LiFTinG and assuming that freeriders keep their probability of being expelled lower than 50%, the performance is close to the baseline.

In this context, LiFTinG incurs a maximum network overhead of only 8%. When freeriders decrease their contribution by 30%, LiFTinG detects 86% of the freeriders and wrongly expels 12% of honest nodes, after only 30 seconds. Most of wrongly expelled nodes deserve it, in a sense, as their actual contribution is smaller than required. However, this is due to poor capabilities, as opposed to freeriders that deliberately decrease their contribution.
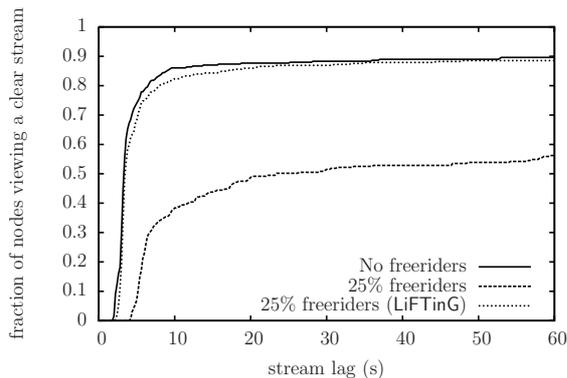

Fig. 1. System efficiency in the presence of freeriders.

Gossip protocols are almost not impacted by crashes [6, 16]. However, high-bandwidth content dissemination with gossip clearly suffers more from freeriders than from crashes. When content is pushed in a single phase, a freerider is equivalent to a crashed node. In three-phase protocols, crashed nodes do not provide upload bandwidth anymore but they do not consume any bandwidth either, as they do not request content from proposers after they crash. On the contrary, freeriders decrease their contribution, yet keep requesting content.

The rest of the paper is organized as follows. Section 2 describes our illustrative gossip protocol and Section 3 lists and classifies the opportunities for nodes to freeride. Section 4 presents LiFTinG and Section 5 formally analyzes its performance backed up by extensive simulations. Section 6 reports on the deployment of LiFTinG over the PlanetLab testbed. Section 7 reviews related work. Section 8 concludes the paper.

2 Three-phase Gossip Protocol

We consider a system of n nodes that communicate over lossy links (e.g., UDP) and can receive incoming data from any other node in the system (i.e., the nodes are not guarded/firewalled, or there exists a means to circumvent such protections [17]). In addition we assume that nodes can pick uniformly at random a set of nodes in the system. This is usually achieved using full membership or a random peer sampling protocol [14,18]. Such sampling protocols can be made robust to Byzantine attacks using techniques such as Brahms [3].

A source broadcasts a stream to all nodes using a three-phase gossip protocol [5,7]. The content is split into multiple chunks uniquely identified by ids. In short, each node periodically proposes a set of chunks it received to a set of random nodes. Upon reception of a proposal, a node requests the chunks it needs and the sender then serves them. All messages are sent over UDP. The three phases are illustrated in Figure 2b.
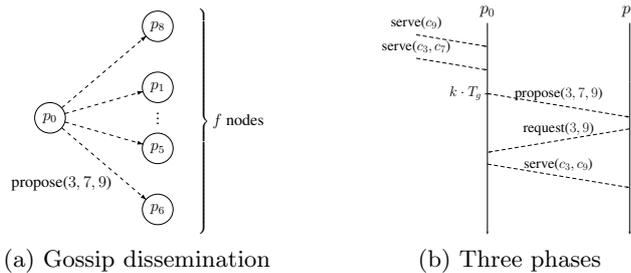

(a) Gossip dissemination   (b) Three phases

Fig. 2. Three-phase generic gossip.

Propose phase A node periodically, i.e., at every gossip period T_g, picks uniformly at random a set of f nodes and proposes to them (as depicted in Figure 2a) the set $\mathcal{P}$ of chunks it received since its last propose phase. The size f of the node set, namely the *fanout*, is the same for all nodes and kept constant over time (the fanout is typically slightly larger than $\ln{(n)}$ [16], that is $f = 12$ for a 10,000-node system). Such a gossip protocol follows an *infect-and-die* process as once a node proposed a chunk to a set of nodes, it does not propose it anymore.

Request phase Upon reception of a proposal of a set $\mathcal{P}$ of chunks, a node determines the subset of chunks $\mathcal{R}$ it needs and requests these chunks.

Serving phase When a proposing node receives a request corresponding to a proposal, it serves the chunks requested. If a request does not correspond to a proposal, it is ignored. Similarly, nodes only serve chunks that were effectively proposed, i.e., chunks in $\mathcal{P} \cap \mathcal{R}$.

3 Freeriding

Nodes are either honest or freeriders. Honest nodes strictly follow the protocol, including the verifications of LiFTinG. Freeriders allow themselves to devi-

ate from the protocol in order to minimize their contribution while maximizing their benefit. In addition, freeriders may adopt any behavior not to be expelled, including lying to verifications, or covering up colluding freeriders' bad actions. Note that under this model, freeriders do not wrongfully accuse honest nodes. Effectively, making honest nodes expelled *(i)* does not increase the benefit of freeriders, *(ii)* does not prevent them from being detected, i.e., detection is based solely on the suspected node's behavior regardless of other nodes' behaviors (details in Section 5.1), and finally *(iii)* leads to an increased proportion of freeriders, degrading the benefit of all nodes. This phenomenon is known as the *tragedy of the commons* [12]. We denote by m the number of freeriders.

Freeriders may deviate from the gossip protocol in the following ways: *(i)* decrease the number of partners to communicate with, *(ii)* bias the partner selection, *(iii)* drop messages they are supposed to send, or *(iv)* modify the content of the messages they send. In the sequel, we exhaustively list all possible attacks in each phase of the protocol, discuss their motivations and impacts, and then extract and classify those that may increase the individual interest of a freerider or the common interest of colluding freeriders. In the sequel, attacks that require or profit to colluding nodes are denoted with a '⋆'.

3.1 Propose phase

During the first phase, a freerider may *(i)* communicate with less than f nodes, *(ii)* propose less chunks than it should, *(iii)* select as communication partners only a specific subset of nodes, or *(iv)* reduce its proposing rate.

(i) **Decreasing fanout** By proposing chunks to $\hat{f} < f$ nodes per gossip period, the freerider trivially reduces the potential number of requests, and thus the probability of serving chunks. Therefore, its contribution in terms of the amount of data uploaded is decreased.

(ii) **Invalid proposal** A proposal is valid if it contains every chunk received in the last gossip period. Proposing only a subset of the chunks received in the last period obviously decreases the potential number of requested chunks. However, a freerider has no interest in proposing chunks it does not have since, contrarily to TfT-based protocols, uploading chunks to a node does not imply that the latter sends chunks in return. In other words, proposing more (and possibly fake) chunks does not increase the benefit of a node and does thus not need to be considered.

(iii) **Biasing the partners selection (⋆)** Considering a group of colluding nodes, a freerider may want to bias the random selection of nodes to favor its colluding partners, so that the group's benefit increases.

(iv) **Increasing the gossip period** A freerider may increase its gossip period, leading to less frequent proposals advertising more, but "older", chunks per proposal. This implies a decreased interest of the requesting nodes and thus a decreased contribution for the sender. This is due to the fact that an old chunk has a lower probability of being of interest as it becomes more replicated over time.

3.2 Pull request phase

Nodes are expected to request only chunks that they have been proposed. A freerider would increase its benefit by opportunistically requesting extra chunks (even from nodes that did not propose these chunks). The dissemination protocol itself prevents this misbehaving by automatically dropping such requests.

3.3 Serving phase

In the serving phase, freeriders may *(i)* send only a subset of what was requested or *(ii)* send junk. The first obviously decreases the freeriders' contribution as they serve fewer chunks than they are supposed to. However, as we mentioned above, in the considered asymmetric protocol, a freerider has no interest in sending junk data, since it does not receive anything in return.

3.4 Summary

Analyzing the basic gossip protocol in details allowed to identify the possible attacks. Interestingly enough, these attacks share similar aspects and can thus be gathered into three classes that dictate the rationale along which our verification procedures are designed.

The first is *quantitative correctness* that characterizes the fact that a node effectively proposes to the correct number of nodes (f) at the correct rate ($1/T_g$). Assuming this first aspect is verified, two more aspects must be further considered: *causality* that reflects the correctness of the deterministic part of the protocol, i.e., received chunks must be proposed in the next gossip period as depicted in Figure 2b, and *statistical validity* that evaluates the fairness (with respect to the distribution specified by the protocol) in the random selection of communication partners.

4 Lightweight Freerider-Tracking in Gossip

LiFTinG is a Lightweight protocol for Freerider-Tracking in Gossip that encourages nodes, in a coercive way, to contribute their fair share to the system, by means of distributed verifications. LiFTinG consists of *(i) direct* verifications and *(ii) a posteriori* verifications. Verifications, that require more information than what is available at the verifying node and the inspected node, are referred to as *cross-checking*. In order to control the overhead of LiFTinG, the frequency at which such verifications are triggered is controlled by a parameter p_{cc}, as described in Section 4.2. Verifications can either lead to the emission of *blames* or to *expulsion*, depending on the gravity of the misbehavior.

Direct verifications are performed regularly while the protocol is running: the nodes' actions are directly checked. They aim at checking that all chunks requested are served and that all chunks served are further proposed to a correct number of nodes, i.e, they check the *quantitative correctness* and *causality*. Direct verifications are composed of *(i)* direct checking and *(ii)* direct cross-checking.

A posteriori verifications are run sporadically. They require each node to maintain a log of its past interactions, namely a *history*. In practice, a node stores a trace of the events that occurred in the last h seconds, i.e., corresponding to the last $n_h = h/T_g$ gossip periods. The history is audited to check the statistical validity of the random choices made when selecting communication partners, namely *entropic check*. The veracity of the history is verified by cross-checking the involved nodes, namely a posteriori cross-checking.

We present the blaming architecture in Section 4.1 and present direct verifications in Section 4.2. Since freeriders can collude not to be detected, we expose how they can cover up each other's misbehaviors in Section 4.3 and address this in Section 4.4. The different attacks and corresponding verifications are summarized in Table 1.

Attack	Type	Detection
fanout decrease ($\hat{f} < f$)	quantitative	direct cross-check
partial propose ($\mathcal{P}$)	causality	direct cross-check
partial serve ($|\mathcal{S}| < |\mathcal{R}|$)	quantitative	direct check
bias partners selection ($\star$)	entropy	entropic check, *a posteriori* cross-check

Table 1. Summary of attacks and associated verifications.

4.1 Blaming architecture

In LiFTinG, the detection of freeriders is achieved by means of a score assigned to each node. When a node detects that some other node freerides, it emits a blame message containing a *blame value* against the suspected node. Summing up the blame values of a node results in a score. For scores to be meaningful, blames emitted by different verifications should be comparable and homogeneous. In order to collect blames targeting a given node and maintain its score, each node is monitored by a set of other nodes named *managers*, distributed among the participants. Blame messages towards a node are sent to its managers. When a manager detects that the score of a node p it monitors drops beyond a *fixed* threshold (the design choice of using a fixed threshold is explained in Section 5.1), it spreads – through gossip – a revocation message against p making the nodes of the system progressively remove p from their membership. A general overview of the architecture of LiFTinG is given in Figure 3.

The blaming architecture of LiFTinG is built on top of the AVMON [23] monitoring overlay. In AVMON, nodes are assigned a *fixed-size* set of M *random* managers *consistent* over time which make it very appealing in our setting, namely a dynamic peer-to-peer environment subject to churn with possibly colluding nodes. The fact that the number M of managers is constant makes the protocol scalable as the monitoring load at each node is independent of the system size. Randomness prevents colluding freeriders from covering each other up and consistency enables long-term blame history at the managers. The monitoring relationship is based on a hash function and can be advertised in a gossip-fashion by piggybacking node's monitors in the view maintenance messages (e.g., exchanges of local views in the distributed peer-sampling service). Doing so, nodes quickly discover other nodes' managers – and are therefore able to blame them
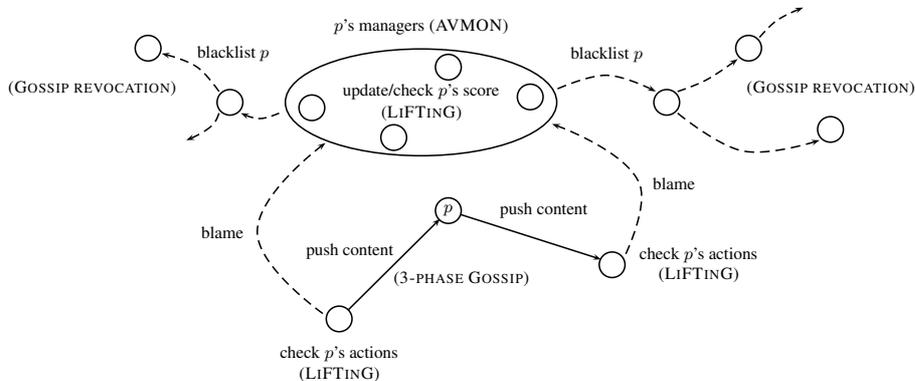
Fig. 3. Overview of LiFTinG

if necessary – even in the presence of churn. In addition, nodes can locally verify (i.e., without the need for extra communication) whether the mapping, node to managers, is correct by hashing the nodes' ip addresses, preventing freeriders from forging fake or colluding managers. In case a manager does not map correctly to a node, a revocation against the concerned node is sent.

4.2 Direct verifications

In LiFTinG, two direct verifications are used. The first aims to ensure that every requested chunk is served, namely a *direct check*. Detection can be done locally and it is therefore always performed. If some requested chunks are missing, the requesting node blames the proposing node by $f/|\mathcal{R}|$ (where $\mathcal{R}$ is the set of requested chunks) for each chunk that has not been delivered.

The second verification checks that received chunks are further proposed to f nodes within the next gossip period. This is achieved by a *cross-checking* procedure that works as follows: a node p_1 that received a chunk c_i from p_0 acknowledges to p_0 that it proposed c_i to a set of f nodes. Then, p_0 sends confirm requests (with probability p_{cc}) to the set of f nodes to check whether they effectively received a propose message from p_1 containing c_i. The f witnesses reply to p_0 with answer messages confirming or not p_1's ack sent to p_0.

Figure 4 depicts the message sequence composing a direct cross-checking verification (with a fanout of 2 for the sake of readability). The blaming mechanism works as follows: *(i)* if the ack message is not received, the verifier p_0 blames the verified node p_1 by f, and *(ii)* for each missing or negative answer message, p_0 blames p_1 by 1.

Since the verification messages (i.e., ack, confirm and confirm responses) for the direct cross-checking are small and in order to limit the subsequent overhead of LiFTinG, direct cross-checking is done exclusively with UDP. The blames corresponding to the different attacks are summarized in Table 2.

Blames emitted by the direct verification procedures of LiFTinG are summed into a score reflecting the nodes' behaviors. For this reason, blame values must be comparable and homogeneous. This means that two misbehaviors that reduce
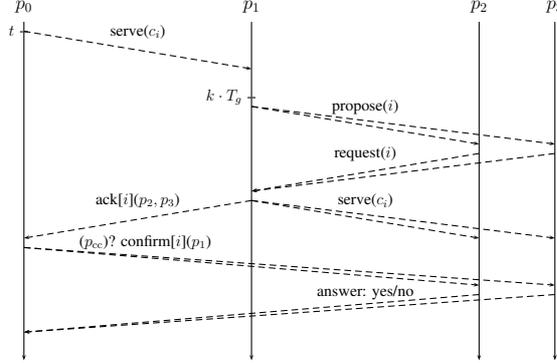
Fig. 4. Cross-checking protocol.

Attacks	Blame values
fanout decrease ($\hat{f} < f$)	$f - \hat{f}$ from each verifier
partial propose	1 (per invalid proposal) from each verifier
partial serve ($|\mathcal{S}| < |\mathcal{R}|$)	$f \cdot (|\mathcal{R}| - |\mathcal{S}|)/|\mathcal{R}|$ from each requester

Table 2. Summary of attacks and associated blame values.

a freerider's contribution by the same amount should lead to the same value of blame, regardless of the misbehaviors and the verification.

We consider a freerider p_f that received c chunks and wants to reduce its contribution by a factor δ ($0 \le \delta \le 1$). To achieve this goal, p_f can: *(i)* propose the c received chunks to only $\hat{f} = (1-\delta) \cdot f$ nodes, *(ii)* propose only a proportion $(1-\delta)$ of the chunks it received, or *(iii)* serve only $(1-\delta) \cdot |\mathcal{R}|$ of the $|\mathcal{R}|$ chunks it was requested. For the sake of simplicity, we assume that $\hat{f}, c \cdot \delta, c/f$ and $\delta \cdot |R|$ are all integers. The number of verifiers, that is, the number of nodes that served the c chunks to p_f is called the *fanin* (f_{in}). On average, we have $f_{in} \simeq f$ and each node serves c/f chunks [9].

We now derive, for each of the three aforementioned misbehaviors, the blame value emitted by the direct verifications.

(i) Fanout decrease (direct cross-check): If p_f proposes all the c chunks to only $\hat{f}$ nodes, it is blamed by 1 by each of the f_{in} verifiers, for each of the $f - \hat{f}$ missing "propose target". This results in a blame value of $f_{in} \cdot (f - \hat{f}) = f_{in} \cdot \delta \cdot f \simeq \delta f^2$.

(ii) Partial propose (direct cross-check): If p_f proposes only $(1-\delta) \cdot c$ chunks to f nodes, it is blamed by f by each of the nodes that provided at least one of the missing chunks. A freerider has therefore interest in removing from its proposal chunks originating from the smallest subset of nodes. In this case, its proposal is invalid from the standpoint of $\delta \cdot f_{in}$ verifiers. This results in a blame value of $\delta \cdot f_{in} \cdot f \simeq \delta \cdot f^2$.

(iii) Partial serve (direct check): If p_f serves only $(1-\delta) \cdot |\mathcal{R}|$ chunks, it is blamed by $f/|\mathcal{R}|$ for each of the $\delta \cdot |\mathcal{R}|$ missing chunks by each of the f requesting nodes. This again results in a blame value of $f \cdot (f/|\mathcal{R}|) \cdot \delta \cdot |\mathcal{R}| = \delta \cdot f^2$.

All misbehaviors lead to the same amount of blame for a given degree of freeriding δ. Therefore, the blame values emitted by the different direct verifications are homogeneous and comparable on average. Thus, they result in a consistent and meaningful score when summed up.

4.3 Fooling the direct cross-check ($\star$)

Considering a set of colluding nodes, nodes may lie to verifications to cover each other up. Consider the situation depicted in Figure 5a, where p_1 is a freerider. If p_0 colludes with p_1, then it will not blame p_1, regardless of p_2's answer. Similarly, if p_2 colludes with p_1, then it will answer to p_0 that p_1 sent a valid proposal, regardless of what p_1 sent. Even when neither p_0 nor p_2 collude with p_1, p_1 can still fool the direct cross-checking thanks to a colluding third party by implementing a *man-in-the-middle attack* as depicted in Figure 5b. Indeed, if a node p_7 colludes with p_1, then p_1 can tell p_0 it sent a proposal to p_7 and tell p_2 that the chunk originated from p_7. Doing this, both p_0 and p_2 will not detect that p_1 sent an invalid proposal. The *a posteriori verifications* presented in the next section address this issue.
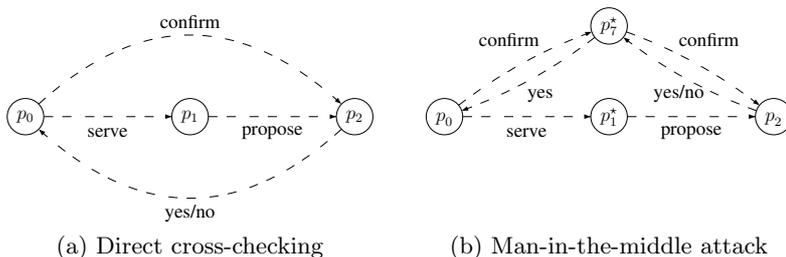

(a) Direct cross-checking   (b) Man-in-the-middle attack

Fig. 5. Direct cross-checking and attack. Colluding nodes are denoted with a '$\star$'.

4.4 A posteriori verifications

As stated in the analysis of the gossip protocol, the random choices made in the partners selection must be checked. In addition, the example described in the previous section, where freeriders collude to circumvent direct cross-checking, highlights the need for statistical verification of a node's past communication partners.

The history of a node that biased its partner selection contains a relatively large proportion of colluding nodes. If only a small fraction of colluding nodes is present in the system, they will appear more frequently than honest nodes in each other's histories and can therefore be detected. Based on this remark, we propose an *entropic check* to detect the bias induced by freeriders on the history of nodes, illustrated in Figure 6.

Every h seconds, each node picks a random node and verifies its local history. When inspecting the history of p, the verifier computes the number of occurrences of each node in the set of proposals sent by p during the last h seconds.
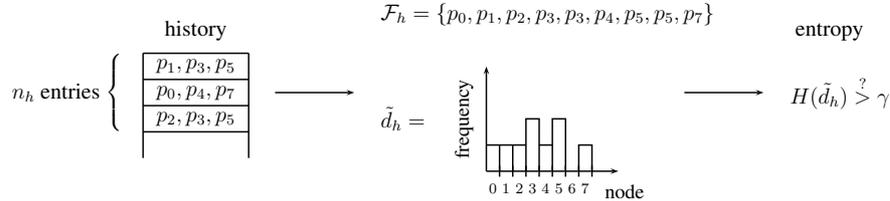
Fig. 6. Entropic check on proposals ($f = 3$).

Defining $\mathcal{F}_h$ as the multiset of nodes to whom p sent a proposal during this period (a node may indeed appear more than once in $\mathcal{F}_h$), the distribution $\tilde{d}_h$ of nodes in $\mathcal{F}_h$ characterizes the randomness of the partners selection. We denote by $\tilde{d}_{h,i}$ the number of occurrences of node i ($i \in \{1, \ldots, n\}$) in $\mathcal{F}_h$ normalized by the size of $\mathcal{F}_h$. Assessing the uniformity of the distribution $\tilde{d}$ of p_1's history is achieved by comparing its Shannon entropy to a threshold γ ($0 \leq \gamma \leq \log_2(n_h f)$).

$$H(\tilde{d}_h) = -\sum_i \tilde{d}_{h,i} \log_2(\tilde{d}_{h,i}) \tag{1}$$

The entropy is maximum when every node of the system appears at most once in $\mathcal{F}_h$ (assuming $n > |\mathcal{F}_h| = n_h f$). In that case, it is equal to $\log_2(n_h f)$. Since the peer selection service may not be perfect, the threshold γ must be tolerant to small deviation with respect to the uniform distribution to avoid *false positives* (i.e., honest nodes being blamed). Details on how to dimension γ are given in Section 5.2.

An entropic check must be coupled with an *a posteriori* cross-checking verification procedure to guarantee the validity of the inspected node's history. Cross-checking is achieved by polling all or a subset of the nodes mentioned in the history for an acknowledgment. The inspected node is blamed by 1 for each proposal in its history that is not acknowledged by the alleged receiver. Therefore, an inspected freerider replacing colluding nodes by honest nodes in its history in order to pass the entropic check will not be covered by the honest nodes and will thus be blamed accordingly.

To cope with potential man-in-the middle attack presented in Section 4.2, a complementary entropic check is performed on the multi-set of nodes $\mathcal{F}'_h$ that asked the nodes in $\mathcal{F}_h$ for a confirmation, i.e., direct cross-checking. On the one hand, for an honest node p_0, $\mathcal{F}'_h$ is composed of the nodes that sent chunks to p_0 – namely its *fanin*. On the other hand, for a freerider $p_0^\star$ that implemented the man-in-the-middle attack, the set $\mathcal{F}'_h$ of $p_0^\star$ contains a large proportion of colluding nodes – the nodes that covered it up for the direct cross-checking – and thus fail the entropic check. If the history of the inspected node does not pass the entropic checks (i.e, fanin and fanout), the node is expelled from the system.

5 Parametrizing LiFTinG

This section provides a methodology to set LiFTinG's parameters. With this aim, the performance of LiFTinG with respect to detection is analyzed theoretically. Closed form expressions of the detection and false positive probabilities function of the system parameters are given. Theoretical results allow the system designer to set the system parameters, e.g., detection thresholds. Theoretical results are obtained by simulations.

This section is split in two. First, the design of the score-based detection mechanism is presented and analyzed taking into account message losses. Second, the entropy-based detection mechanism is analyzed taking into account the underlying peer-sampling service. Both depend on the degree of freeriding and on the favoring factor, i.e., how freeriders favor colluding partners.

5.1 Score-based detection

Due to message losses, a node may be wrongfully blamed, i.e., blamed even though it follows the protocol. Freeriders are additionally blamed for their mis-behaviors. Therefore, the score distribution among the nodes is expected to be a mixture of two components corresponding respectively to those of honest nodes and freeriders. In this setting, likelihood maximization algorithms are tradition-ally used to decide whether a node is a freerider or not. Such algorithms are based on the relative score of the nodes and are thus not sensitive to wrongful blames. Effectively, wrongful blames have the same impact on honest nodes and freeriders.

However, in the presence of freeriders, two problems arise when using relative score-based detection: *(i)* freeriders are able to decrease the probability of being detected by wrongfully blaming honest nodes, and *(ii)* the score of a node joining the system is not comparable to those of the nodes already in the system. For these reasons, in LiFTinG, the impact of wrongful blames, due to message losses, is automatically compensated. Detection thus consists in comparing the nodes' compensated scores to a fixed threshold η. In short, when the compensated score of a node drops below η, the managers of that node broadcast a revocation message expelling the node from the system using gossip.

Considering message losses independently drawn from a Bernoulli distribu-tion of parameter p_l (we denote by $p_r = 1 - p_l$ the probability of reception), we derive a closed-form expression for the expected value of the blames applied to honest nodes by direct verifications during a given timespan. Periodically in-creasing all scores accordingly leads to an average score of 0 for honest nodes. This way, the fixed threshold η can be used to distinguish between honest nodes and freeriders. To this end, we analyze, for each verification, the situations where message losses can cause wrongful blames and evaluate their average impact. For the sake of the analysis, we assume that *(i)* a node receives at least one chunk during every gossip period (and therefore it will send proposals during the next gossip period), and *(ii)* each node requests a constant number $|\mathcal{R}|$ of chunks for

each proposal it receives. We consider the case where cross-checking is always performed, i.e., $p_{cc} = 1$.

Direct check (dc) For each requested chunk that has not been served, the node is blamed by $f/|\mathcal{R}|$. If the proposal is received but the request is lost (i.e., $p_r(1-p_r)$), the node is blamed by f ((a) in Equation 2). Otherwise, when both the proposal and the request message are received (i.e., p_r^2), the node is blamed by $f/|R|$ for each of the chunks lost (i.e., $(1-p_r)|R|$) ((b) in Equation 2). The expected blame applied to an honest node (by its f partners), during one gossip period, due to message losses is therefore:

$$\tilde{b}_{dc} = f \cdot \left[\overbrace{p_r(1-p_r) \cdot f}^{(a)} + \overbrace{p_r^2 \cdot (1-p_r)|\mathcal{R}| \cdot \frac{f}{|\mathcal{R}|}}^{(b)} \right] = p_r(1-p_r^2) \cdot f^2 \quad (2)$$

Direct cross-checking (dcc) On average, a node receives f proposals during each gossip period. Therefore a node is subject to f direct cross-checking verifications and each verifier asks for a confirmation to the f partners of the inspected node. Let p_1 be the inspected node and p_0 a verifier. First, note that p_0 verifies p_1 only if it served chunks to p_1, which requires that its proposal and the associated request have been received (i.e., p_r^2). If at least one chunk served by p_0 or the ack has been lost (i.e., $1 - p_r^{|\mathcal{R}|+1}$), p_0 will blame p_1 by f regardless of what happens next, since all the f proposals sent by p_1 are invalid from the standpoint of p_0 ((a) in Equation 3). Otherwise, that is, if all the chunks served and the ack have been received (i.e., $p_r^{|\mathcal{R}|+1}$), p_0 blames p_1 by 1 for each negative or missing answer from the f partners of p_1. This situation occurs when the proposal sent by p_1 to a partner, the confirm message or the answer is lost (i.e., $1 - p_r^3$) ((b) in Equation 3).

$$\tilde{b}_{cc} = f \cdot p_r^2 \left[\overbrace{(1-p_r^{|R|+1}) \cdot f}^{(a)} + \overbrace{f \cdot p_r^{|\mathcal{R}|+1}(1-p_r^3)}^{(b)} \right] = p_r^2(1-p_r^{|\mathcal{R}|+4}) \cdot f^2 \quad (3)$$

From the previous analysis, we obtain a closed-form expression for the expected value of the blame b applied to an honest node by direct verifications due to message losses:

$$\tilde{b} = \tilde{b}_{dc} + \tilde{b}_{cc} = p_r(1 + p_r - p_r^2 - p_r^{|\mathcal{R}|+5}) \cdot f^2 \quad . \quad (4)$$

The blame value b' applied to a freerider by direct verifications depends on its *degree of freeriding* Δ that characterizes its deviation to the protocol. Formally, we define the degree of freeriding as a 3-uple $\Delta = (\delta_1, \delta_2, \delta_3)$, $0 \le \delta_1, \delta_2, \delta_3 \le 1$, so that a freerider contacts only $(1 - \delta_1) \cdot f$ nodes per gossip period, proposes the chunks received from a proportion $(1 - \delta_2)$ of the nodes that served it in the previous gossip period, and serves $(1 - \delta_3) \cdot |\mathcal{R}|$ chunks to each requesting node. With the same methodology as for $\tilde{b}$, we get:

$$\tilde{b}'(\Delta) = (1 - \delta_1) \cdot p_r \left(1 - p_r^2(1 - \delta_3) \right) \cdot f^2 + \delta_2 \cdot f^2 + \quad (5)$$
$$(1 - \delta_2) \cdot p_r^2 \cdot \left[p_r^{|\mathcal{R}|+1}(1 - p_r^3(1 - \delta_1)) + (1 - p_r^{|\mathcal{R}|+1}) \right] \cdot f^2$$

Note that the gain in terms of the upload bandwidth saved by a freerider is $1 - (1 - \delta_1)(1 - \delta_2)(1 - \delta_3)$. Following the same line of reasoning, a closed-form expression of the standard deviation $\sigma(b)$ (resp. $\sigma(b'(\Delta))$) of b (resp. $b'(\Delta)$) can be derived.

Figure 7 depicts the distribution of compensated and normalized scores (see Formula 6) in the presence of $1,000$ freeriders of degree $\delta = \delta_1 = \delta_2 = \delta_3 = 0.1$ in a $10,000$-node system after $r = 50$ gossip periods. The message loss rate is set to 7%, the fanout f to 12 and $|\mathcal{R}| = 4$. The scores of the nodes have been increased by $-\tilde{b} = 72.95$, according to Formula (4). We plot separately the distribution of scores among honest nodes and freeriders. As expected, the probability density function (Figure 7a) is split into two disjoint modes separated by a gap: the lowest (i.e., left most) mode corresponds to freeriders and the highest one to honest nodes. We observe that the average score (dotted line) is close to zero (< 0.01) which means that the wrongful blames have been successfully compensated.
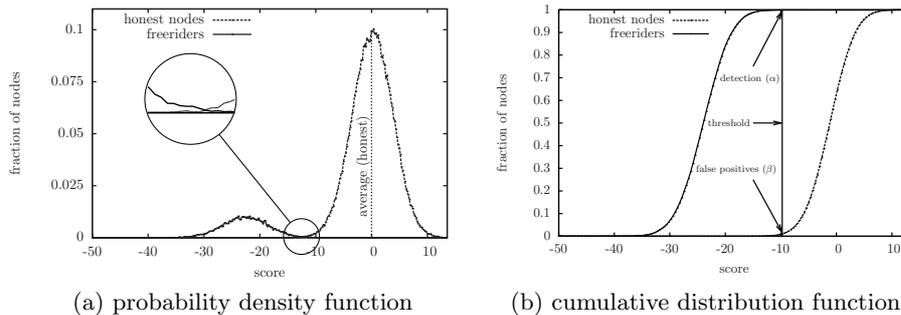

(a) probability density function   (b) cumulative distribution function

Fig. 7. Distribution of normalized scores in the presence of freeriders ($\delta = 0.1$).

We now evaluate the ability of LiFTinG to detect freeriders (probability of detection α) and the proportion of honest nodes wrongfully expelled from the system (probability of false positives β). Figure 7b depicts the cumulative distribution function of scores and illustrates the notion of detection and false positives for a given value of the detection threshold.

In order to enable the use of a fixed threshold η, the scores are compensated with respect to message losses and normalized by the number of gossip periods r the node spent in the system. At the t-th gossip period, the score of a node writes:

$$s = -\frac{1}{r} \sum_{i=0}^{r} (b_{t-i} - \tilde{b}), \qquad (6)$$

where b_i is the value of the blames applied to the node during the i-th gossip period. From the previous analysis, we get the expectation and the standard deviation of the blames applied to honest nodes at each round due to message losses, therefore, assuming that the b_i are independent and identically distributed (i.i.d.), we get $\mathbb{E}[s] = 0$ and $\sigma(s) = \sigma(b)/\sqrt{r}$. Using Bienaymé-Tchebychev's

inequality we get:

$$\beta = P\left(s < \eta\right) \leq \frac{\sigma(b)^2}{r \cdot \eta^2} \qquad \text{and} \qquad \alpha \geq 1 - \frac{\sigma(b'(\Delta))^2}{r \cdot (\tilde{b}'(\Delta) - \eta)^2} \qquad (7)$$

We set the detection threshold η to -9.75 so that the probability of false positive is lower than 1%, we assume that freeriders perform all possible attacks with degree δ and we observe the proportion of freeriders detected by LiFTinG for several values of δ. Figure 8 plots α function of δ. We observe that a node freeriding by 5% is detected with probability 0.65. Beyond 10% of freeriding, a node is detected over 99% of the time. It is commonly assumed that users are willing to use a modified version of the client application only if it increases significantly their benefit (resp. decreases their contribution). In FlightPath [21], this threshold is assumed to be around 10%. With LiFTinG, a freerider achieves a gain of 10% for $\delta = 0.035$ which corresponds to a probability of being detected of 50% (Figure 8).
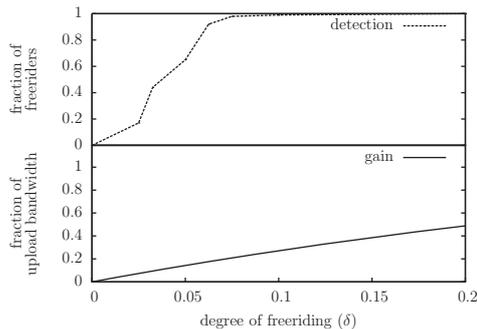

Fig. 8. Proportion of freeriders detected by LiFTinG.

5.2 Entropy-based detection

For the sake of fairness and in order to prevent colluding nodes from covering each other up, LiFTinG includes an entropic check assessing the statistical validity of the partner selection. To this end, the entropy H of the distribution of the inspected node's former partners is compared to a threshold γ. The distribution of the entropy of honest nodes' histories depends on the peer sampling algorithm used and the random numbers generator. It can be estimated by simulations. Figure 9a depicts the distribution of entropy for a history of $n_h f = 600$ partners ($n_h = 50$ and $f = 12$) of a 10,000-node system using a full membership-based partner selection. The observed entropy ranges from 9.11 to 9.21 for a maximum reachable value of $\log_2\left(n_h f\right) = 9.23$. Similarly, the entropy of the fanin multiset $\mathcal{F}'_h$, i.e., nodes that selected the inspected node as partner, is depicted in Figure 9b. The observed entropy ranges from 8.98 to 9.34.

With $\gamma = 8.95$ the probability of wrongfully expelling a node during local auditing is negligible.
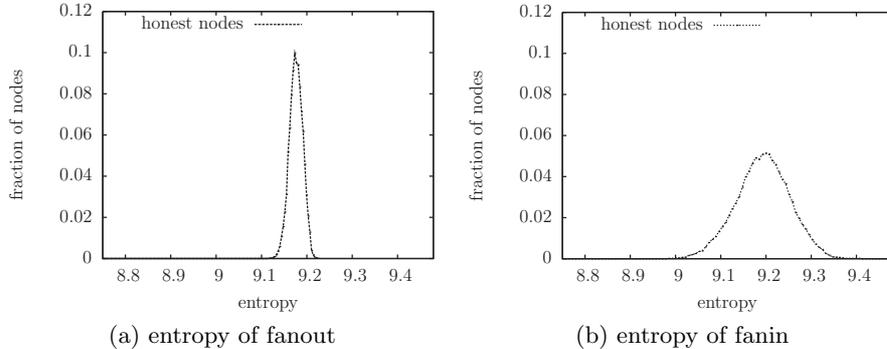
Fig. 9. Entropy distribution of the nodes' histories using a full membership.

We now analytically determine to what extent a freerider can bias its partner selection without being detected by local auditing, given a threshold γ and a number of colluding nodes m'. Consider a freerider that biases partner selection in order to favor colluding freeriders by picking a freerider as partner with probability p_m and an honest node with probability $1 - p_m$. We seek the maximum value of p_m a freerider can use without being detected, function of γ and m'.

It can be proved [10] that, to maximize the entropy of its history, a freerider must choose uniformly at random its partners in the chosen class, i.e., honest or colluding. In that case, the entropy of its history writes (for $m' < n_h f$):

$$H(\mathcal{F}_h) = -p_m \log_2 \left(\frac{p_m}{m'} \right) - (1 - p_m) \log_2 \left(\frac{1}{n_h \cdot f} \right) \qquad (8)$$

Inverting numerically Formula (8), we deduce that for $\gamma = 8.95$ a freerider colluding with 25 other nodes can serve its colluding partners up to 15% of the time, without being detected.

6   Evaluation and Experimental Results

We now evaluate LiFTinG on top of the gossip-based streaming protocol described in [7], over the PlanetLab testbed. We describe the experimental setup in Section 6.1. We evaluate the performance of LiFTinG showing its small overhead in Section 6.2 and its precision and speed at detecting freeriders in Section 6.3.

6.1   Experimental setup

We have deployed and executed LiFTinG on a 300 PlanetLab node testbed, broadcasting a stream of 674 kbps in the presence of 10% of freeriders. The freeriders *(i)* contact only $\hat{f} = 6$ random partners ($\delta_1 = 1/7$), *(ii)* propose only 90% of what they receive ($\delta_2 = 0.1$) and finally *(iii)* serve only 90% of what they are requested ($\delta_3 = 0.1$). The fanout of all nodes is set to 7 and the gossip period is set to 500 ms. The blaming architecture uses $M = 25$ managers for each node.

6.2 Practical cost

We report on the overhead measurements of direct and *a posteriori* verifications (including blame messages sent to the managers) for different stream rates.

Direct verifications. Table 3 gives the bandwidth overhead of the direct verifications of LiFTinG for three values of p_{cc}. Note that the overhead is not null when $p_{\mathrm{cc}} = 0$ since ack messages are always sent. Yet, we observe that the overhead is negligible when $p_{\mathrm{cc}} = 0$ (i.e., when the system is healthy) and remains reasonable when $p_{\mathrm{cc}} = 1$ (i.e., when the system needs to be purged from freeriders).

	direct verifications			a posteriori verifications
	$p_{cc} = 0$	$p_{cc} = 0.5$	$p_{cc} = 1$	
674 kbps stream	1.07%	4.53%	8.01%	3.60%
1082 kbps stream	0.69%	3.51%	5.04%	2.89%
2036 kbps stream	0.38%	2.80%	2.76%	1.74%

Table 3. Practical overhead

A posteriori verifications. A history message contains n_h entries. Each entry consists of f nodes identifiers and the chunk ids that were proposed. Both the fanout and fanin histories are sent upon a posteriori verification.

Besides the entropic checks, *a posteriori* cross-checking is performed on a subset of the fanout or fanin entries. We measured the maximum overhead, that is when the whole fanout and fanin histories are cross-checked. The overhead incurred by *a posteriori* verifications in our experimental setup (i.e., a history size $n_h = 50$, a gossip period of 500 milliseconds, a fanout of $f = 7$ and *a posteriori* verification period of $h = 25$ seconds) is given in Table 3.

6.3 Experimental results

We have executed LiFTinG with $p_{\mathrm{cc}} = 1$ and $p_{\mathrm{cc}} = 0.5$. Figure 10 depicts the scores obtained after 25, 30 and 35 seconds when running direct verifications and cross-checking. The scores have been compensated as explained in the analysis, assuming a loss rate of 4% (average value for UDP packets observed on PlanetLab).

The two cumulative distribution functions for honest nodes and freeriders are clearly separated. The threshold for expelling freeriders is set to -9.75 (as specified in the analysis). In Figure 10b ($p_{\mathrm{cc}} = 1$, after 30 s) the detection mechanism expels 86% of the freeriders and 12% of the honest nodes. In other words, after 30 seconds, 14% of freeriders are not yet detected and 12% represent false positives, mainly corresponding to honest nodes that suffer from very poor connection (e.g., limited connectivity, message losses and bandwidth limitation). These nodes do not deliberately freeride, but their connection does not allow them to contribute their fair share. This is acceptable as such nodes should not have been allowed to join the system in the first place. As expected, with p_{cc} set to 0.5 the detection is slower but not twice as slow. Effectively, with nodes freeriding with $\delta_3 > 0$ (i.e., partial serves) the direct checking blames freeriders
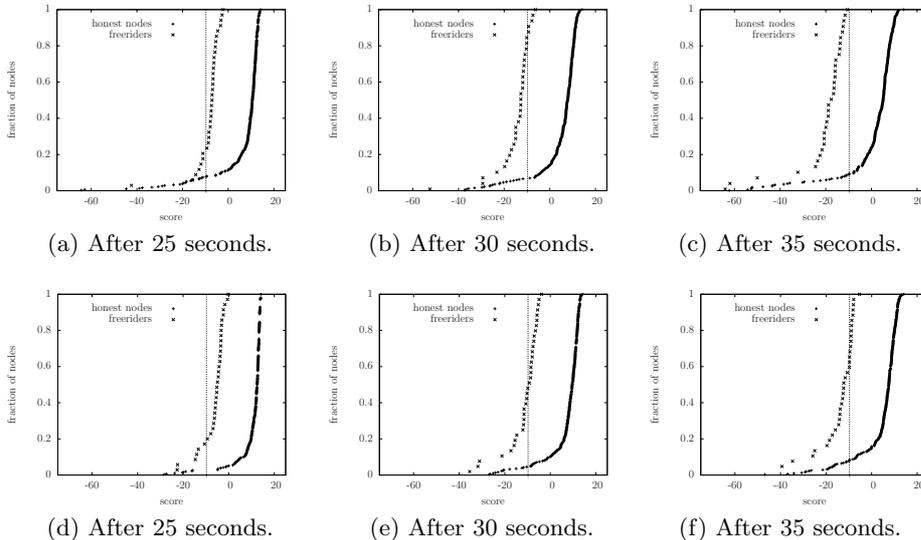
Fig. 10. Scores CDF with $p_{cc} = 1$ (above) and $p_{cc} = 0.5$ (below).

without the need for any cross-check. This explains why the detection after only 35 seconds with $p_{cc} = 0.5$ (Figure 10f) is comparable to the detection after 30 seconds with $p_{cc} = 1$ (Figure 10b).

As stated in the analysis, we observe that the gap between the two cumulative distribution functions widens over time. However, the variance of the score does not decrease over time (for both honest nodes and freeriders). This is due to the fact that we considered in the analysis that the blames applied to a given node during distinct gossip periods were independent and identically distributed (i.i.d.). In practice however, successive gossip periods are correlated. Effectively, a node with a poor connection is usually blamed more than nodes with high capabilities, and this remains true over the whole experiment.

7 Related Work

TfT distributed incentives have been broadly used to deal with freeriders in file sharing systems based on symmetric exchanges, such as BitTorrent [4]. However, there is a number of attacks, mainly targeting the opportunistic unchoking mechanism (i.e., asymmetric push), allowing freeriders to download contents with no or a very small contribution [22, 24].

FlightPath (built on top of BAR Gossip) [21] is a gossip-based streaming application that fights against freeriding using verifications on partner selection and chunk exchanges. FlightPath operates in a gossip fashion for partner selection and is composed of opportunistic pushes performed by altruistic nodes (essential for the efficiency of the protocol) and balanced pairwise exchanges secured by TfT. Randomness of partner selection is verified by means of a pseudo-random number generator with signed seeds, and symmetric exchanges

are made robust using cryptographic primitives. FlightPath prevents attacks on opportunistic pushes by turning them into symmetric exchanges: each peer must reciprocate with junk chunks when opportunistically unchoked. This results in a non-negligible waste of bandwidth. It is further demonstrated in [13] that BAR Gossip presents scalability issues, not to mention the overhead of cryptography.

PeerReview [11] deals with malicious nodes following an accountability approach. Peers maintain signed logs of their actions that can be checked using a reference implementation running in addition to the application. When combined with CSAR [2], PeerReview can be applied to non-deterministic protocols. However, the intensive use of cryptography and the sizes of the logs maintained and exchanged drastically reduce the scalability of this solution. In addition, the validity of PeerReview relies on the fact that messages are always received which is not the case over the Internet.

The two approaches that relate the most to LiFTinG are the distributed auditing protocol proposed in [13] and the passive monitoring protocol proposed in [15]. In the first protocol, freeriders are detected by cross-checking their neighbors' reports. The latter focuses on gossip-based search in the Gnutella network. The peers monitor the way their neighbors forward/answer queries in order to detect freeriders and query droppers. Yet, contrarily to LiFTinG – which is based on random peer selection – in both protocols the peers's neighborhoods are static, forming a fixed mesh overlay. These techniques thus cannot be applied to gossip protocols. In addition, the situation where colluding peers cover each other up (not addressed in the papers) makes such monitoring protocols vain.

8   Conclusion

We presented LiFTinG, a protocol for tracking freeriders in gossip-based asymmetric data dissemination systems. Beyond the fact that LiFTinG deals with the randomness of the protocol, LiFTinG precisely relies on this randomness to robustify its verification mechanisms against colluding freeriders with a very low overhead. We provided a theoretical analysis of LiFTinG that allows system designers to set its parameters to their optimal values and characterizes its performance backed up by extensive simulations. We reported on our experimentations on PlanetLab which prove the practicability and efficiency of LiFTinG.

We believe that, beyond gossip protocols, LiFTinG can be used to secure the asymmetric component of TfT-based protocols, namely *opportunistic unchoking*, which is considered to constitute their Achilles heel [22, 24].

References

1. E. Adar and B. Huberman. Free riding on Gnutella. *First Monday*, 5, 2000.
2. M. Backes, P. Druschel, A. Haeberlen, and D. Unruh. CSAR: A Practical and Provable Technique to Make Randomized Systems Accountable. In *NDSS*, 2009.
3. E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer. Brahms: Byzantine Resilient Random Membership Sampling. *Computer Networks*, 53:2340–2359, 2009.

4. B. Cohen. Incentives Build Robustness in BitTorrent. In *P2P Econ*, 2003.
5. M. Deshpande, B. Xing, I. Lazardis, B. Hore, N. Venkatasubramanian, and S. Mehrotra. CREW: A Gossip-based Flash-Dissemination System. In *ICDCS*, 2006.
6. P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight Probabilistic Broadcast. *TOCS*, 21:341–374, 2003.
7. D. Frey, R. Guerraoui, A.-M. Kermarrec, M. Monod, and V. Quéma. Stretching Gossip with Live Streaming. In *DSN*, 2009.
8. D. Frey, R. Guerraoui, B. Koldehofe, A.-M. Kermarrec, M. Mogensen, M. Monod, and V. Quéma. Heterogeneous Gossiping. In *Middleware*, 2009.
9. A. Ganesh, A.-M. Kermarrec, and L. Massoulié. SCAMP: Peer-to-peer Lightweight Membership Service for Large-scale Group Communication. In *NGC*, 2001.
10. R. Guerraoui, K. Huguenin, A.-M. Kermarrec, and M. Monod. LiFT: Lightweight Freerider-Tracking Protocol. Research Report RR-6913, INRIA, 2009.
11. A. Haeberlen, P. Kouznetsov, and P. Druschel. PeerReview: Practical Accountability for Distributed Systems. In *SOSP*, 2007.
12. G. Hardin. The Tragedy of the Commons. *Science*, 162:1243–1248, 1968.
13. M. Haridasan, I. Jansch-Porto, and R. Van Renesse. Enforcing Fairness in a Live-Streaming System. In *MMCN*, 2008.
14. M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based Peer Sampling. *TOCS*, 25:1–36, 2007.
15. M. Karakaya, I. Körpeoğlu, and O. Ulusoy. Counteracting Free-riding in Peer-to-Peer Networks. *Computer Networks*, 52:675–694, 2008.
16. A.-M. Kermarrec, L. Massoulié, and A. Ganesh. Probabilistic Reliable Dissemination in Large-Scale Systems. *TPDS*, 14:248–258, 2003.
17. A.-M. Kermarrec, A. Pace, V. Quéma, and V. Schiavoni. NAT-resilient Gossip Peer Sampling. In *ICDCS*, 2009.
18. V. King and J. Saia. Choosing a Random Peer. In *PODC*, 2004.
19. R. Krishnan, M. Smith, Z. Tang, and R. Telang. The Impact of Free-Riding on Peer-to-Peer Networks. In *HICSS*, 2004.
20. B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, and X. Zhang. Inside the New Coolstreaming: Principles, Measurements and Performance Implications. In *INFOCOM*, 2008.
21. H. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robinson, L. Alvisi, and M. Dahlin. FlightPath: Obedience vs Choice in Cooperative Services. In *OSDI*, 2008.
22. T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free Riding in BitTorrent is Cheap. In *HotNets*, 2006.
23. R. Morales and I. Gupta. AVMON: Optimal and Scalable Discovery of Consistent Availability Monitoring Overlays for Distributed Systems. *TPDS*, 20:446–459, 2009.
24. M. Sirivianos, J. Park, R. Chen, and X. Yang. Free-riding in BitTorrent with the Large View Exploit. In *IPTPS*, 2007.
25. V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast. In *ICNP*, 2006.
26. M. Zhang, Q. Zhang, L. Sun, and S. Yang. Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better? *JSAC*, 25:1678–1694, 2007.