LiFTinG: Lightweight Freerider-Tracking in Gossip

Rachid Guerraoui
EPFL
rachid.guerraoui@epfl.ch

Kévin Huguenin
Université de Rennes 1 / IRISA
kevin.huguenin@irisa.fr

Anne-Marie Kermarrec
INRIA Rennes – Bretagne Atlantique
anne-marie.kermarrec@inria.fr

Maxime Monod
EPFL
maxime.monod@epfl.ch

Swagatika Prusty
IIT Guwahati
swagatika@iitg.ernet.in

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 each other up.

We present extensive analytical evaluations of LiFTinG, backed up by simulations and PlanetLab experiments. In a 300-node system, where a stream of 674 kbps is broadcast, LiFTinG incurs a maximum overhead of only 8%. 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 [3, 5, 6]. In such *asymmetric* systems, nodes push packet identifiers to a dynamically changing random subset of other nodes, who subsequently pull packets of interest.

The efficiency of such protocols highly rely 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 could 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 such as streaming. The impact of freeriders is even more critical when they collude, i.e., they collaborate to decrease their individual and common contribution to the system.

In systems where exchanges are *symmetric*, one of the most popular mechanisms to address freeriding is the celebrated Tit-for-Tat (TfT). TfT forces nodes to collaborate by enforcing the benefit of a node to be directly proportional to its contribution [4]. BarGossip [21] applies TfT in gossip protocols by forcing symmetric exchanges. In this context, freeriders can be modeled as *rational* nodes that contribute to symmetric exchanges by the minimum amount required to get the maximum benefit in return [21].

In *asymmetric* systems [3, 5, 20, 25] where nodes altruistically push content to other nodes, i.e., without asking anything in return, the benefit of a node is not directly correlated to its contribution but rather to the global *health* of the system. However, a strong correlation between the contribution of a node and its benefit can be artificially established, in a coercive way, by means of verification mechanisms that expel the nodes which do not contribute their fair share. In this case, freeriders can be modeled as *wise* nodes that decrease their contribution as much as possible while keeping the probability of being expelled low.

We consider a generic gossip protocol where data is disseminated following an asymmetric push scheme where TfT cannot be used. In this context, we propose LiFTinG a coercive lightweight mechanism to track freeriders in gossip protocols. To our knowledge, LiFTinG is the first protocol to secure asymmetric gossip protocols against possibly colluding freeriders. At the core of LiFTinG is 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 short delay) by cross-checking nodes' logs. Statistical procedures check that the interactions of a node are evenly dis-

tributed in the system using statistical techniques. Interestingly enough, the high dynamic and strong randomness of gossip-based protocols that may be considered as a difficulty at a first glance, happens to help tracking freeriders. Effectively, LiFTinG exploits the fact that nodes pick neighbors at random to prevent collusion: the fact that a node interacts with a large subset of the nodes, chosen at random, drastically limits its possibility to freeride without being detected, as it prevents it from continuously and deterministically choosing colluding partners that would cover it up.

LiFTinG is lightweight as it does not use heavyweight cryptography and incurs only a very low overhead in terms of bandwidth. This overhead can be dynamically adjusted and potentially reduced to zero when the system is healthy. In addition, LiFTinG is fully decentralized and evenly distributed among the nodes. Finally, LiFTinG provides a good probability of detecting freeriders while keeping the probability of false positives (i.e., inaccurately classifying a correct node as a freerider) very low.

Figure 1 depicts the health (i.e., the proportion of nodes able to view the stream as a function of the stream lag) in a 300 PlanetLab-node system where a stream of 674 kbps is broadcast. We consider wise freeriders that decrease their contribution as much as possible while keeping the probability of being caught lower than 50%. In short, without LiFTinG, the system collapses in the presence of freeriders while the performance remains very close to the baseline with LiFTinG.

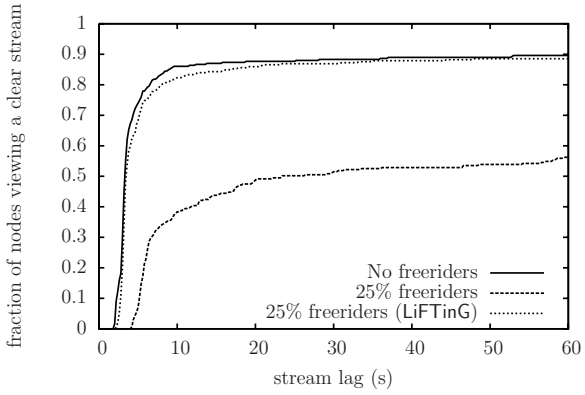


Figure 1. System efficiency in the presence of freeriders.

In a 300-node system deployed over PlanetLab, where a stream of 674 kbps is broadcast, LiFTinG incurs a maximum overhead of only 8%. With 10% of freeriders decreasing their contribution by 30%, LiFTinG detects 86% of the freeriders and wrongly expels 12% of honest nodes (most of them being nodes whose decreased contribution is due to poor capabilities: these nodes deserve, in a sense, to be expelled as well as freeriders) after only 30 seconds.

We believe that, beyond gossip protocols, LiFTinG can be used to secure the asymmetric component of TfT-based protocols, namely *opportunistic unchoking*. This is considered to constitute their Achile's heel [23, 24]. We can consider for instance a gossip protocol, secured by LiFTinG, to disseminate fresh chunks in the system, coupled with a protocol based on symmetric exchanges to complete the dissemination using traditional swarming and TfT.

The rest of the paper is organized as follows. Section 2 defines the system model. Section 3 describes our illustrative gossip protocol and Section 4 lists and classifies the opportunities for nodes to freeride. Section 5 presents LiFTinG and Section 6 formally analyzes its performance backed up by extensive simulations. Section 7 reports on the deployment of LiFTinG over the PlanetLab testbed. Section 8 reviews related work. Section 9 concludes the paper.

2. System model

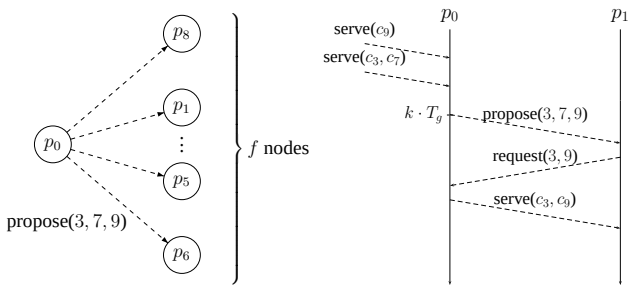
We consider a system of n nodes that communicate over lossy links (e.g., UDP) and can receive incoming data from any other node in the system (i.e., the nodes are not guarded/firewalled, or there exists a means to circumvent such protections [17]). In addition we assume that nodes can pick uniformly at random a set of nodes in the system. This is usually achieved using full membership or a random peer sampling protocol [13, 18].

Nodes are either honest or freeriders. Honest nodes strictly follow the protocol (including verifications) while freeriders allow themselves to deviate from the protocol in order to minimize their contribution. Colluding freeriders want to minimize their contribution and maximize the benefit of the coalition. In addition, they may lie not to be detected or even cover colluding freeriders up. However, they are *wise* as they behave in such a way that the probability to be expelled remains low. Freeriders do not wrongfully accuse honest nodes. Effectively, making honest nodes expelled *(i)* does not increase their benefit and *(ii)* leads to an increased proportion of freeriders, thus degrading the overall contribution of the system that results, in the end, in degrading all nodes' benefit. This phenomenon, known as the *tragedy of the commons* [10] decreases the overall benefit of the system, including freeriders. We denote by m the number of freeriders in the system.

3. Basic gossip protocol

We consider a system where content is broadcast from a source to all nodes using a three-phase gossip-based protocol [5, 6]. The content is split into multiple chunks that are identified by chunk ids. In short, each node periodically proposes a set of chunks to a set of random nodes (i.e., proposing the chunk ids). Upon reception of a proposal, a node requests the sender the chunk ids it needs and the sender then serves the requested chunks. All messages are sent over UDP. The three phases are illustrated in Figure 2b.

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



(a) Gossip dissemination (b) Three phases

Figure 2. Three-phase generic gossip.

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 the sender to serve 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 $\mathcal{P} \cap \mathcal{R}$).

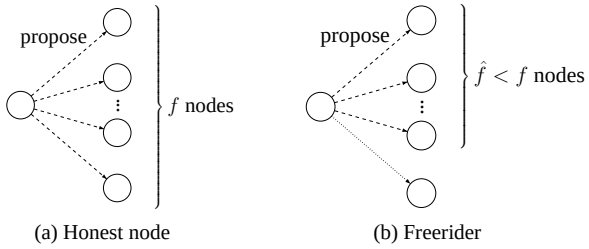
4. Freeriding

Freeriders may deviate from the protocol in three ways: *(i)* bias the partner selection, *(ii)* drop messages they are supposed to send, or *(iii)* modify the content of the messages they send. 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 serve colluding nodes are denoted with a '$\star$'. This analysis of the three-phase protocol is at the core of our lightweight freerider tracking scheme – LiFTinG.

4.1 Propose phase

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

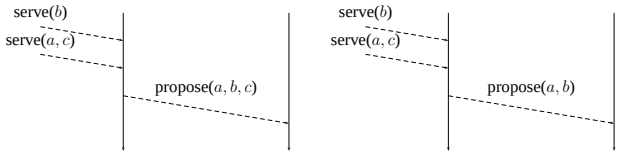
(i) **Decreasing fanout** By communicating to $\hat{f} < f$ nodes, the freerider trivially reduces the potential number of requests, and thus the probability of serving chunks. Therefore, its contribution in terms of the amount of data uploaded is decreased while still fulfilling the $\hat{f}$ received requests as illustrated in Figure 3.



(a) Honest node (b) Freerider

Figure 3. A freerider communicates with $\hat{f} < f$ partners.

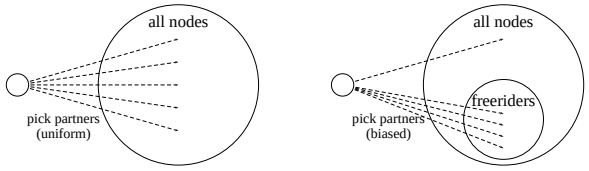
(ii) **Invalid proposal** A proposal is valid if it contains every chunk received in the last gossip period. Proposing only a subset of the chunks received in the last period, as illustrated in Figure 4, obviously decreases the number of requested chunks. However, a freerider has no interest in proposing 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.



(a) Honest node (b) Freerider

Figure 4. A freerider deliberately removes some chunks (c here) from its proposal.

(iii) **Biasing the partners selection ($\star$)** Considering a group of colluding nodes, a freerider may want to bias the random selection of nodes to privilege its colluding partners, so that the group's benefit increases, as illustrated in Figure 5.

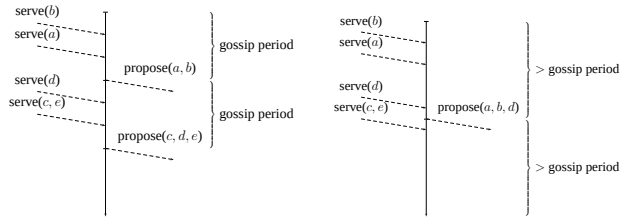


(a) Honest node (b) Colluding freeriders

Figure 5. An honest node picks communication partners uniformly at random from the set of all nodes whereas a freerider biases the partner selection to pick mainly colluding nodes.

(iv) **Increasing the gossip period** A freerider may increase its gossip period, leading to less frequent proposals, i.e., advertising more chunks per proposal, but "older" ones, as illustrated in Figure 6. 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 to be of interest as it becomes more replicated over time.



(a) Honest node (b) Freerider

Figure 6. With a larger gossip period, some proposed chunks are unlikely to be requested (e.g., a and b here).

4.2 Pull request phase

Nodes are expected to request only chunks 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 as described above.

4.3 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 less 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.

Analyzing the basic gossip protocol (Section 3) 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.

5. 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 lightweight verifications. LiFTinG consists of two kinds of verifications: *(i) direct* verifications and *(ii) a posteriori* auditing procedures. Verifications can either lead to the emission of *blames* or to *expulsion* depending on the gravity of the detected misbehavior. The value of a blame is proportional to the number of invalid pushes (i.e., proposal, request and serves). Therefore, the blames emitted by the verification procedures are directly comparable and can thus be summed up into a consistent reputation score.

Direct verifications 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., it checks the *quantitative correctness* and *causality*. Direct verifications are performed by nodes and are triggered with probability p_{dcc}: it can be triggered at each serve ($p_{dcc} = 1$), never ($p_{dcc} = 0$) if the system is considered healthy, or anything in between.

The a posteriori auditing procedures are run sporadically and consist in asking a suspected node for a log of its past interactions to check the *statistical validity* of the random choices made when selecting communication partners. To this aim, every node logs a bounded-size *history* of sent and received messages. More specifically, each node maintains 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.

5.1 Blaming architecture

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. We use an underlying distributed reputation mechanism allowing nodes to blame other nodes and to access each other's scores in a decentralized fashion. This is achieved using an Alliatrust-like architecture [7]. In Alliatrust, each node is assigned M random managers that store a copy of its reputation score. When a node wants to know the score of a node p, it contacts p's managers and votes over the M returned values. In order to be resilient to message losses and malicious attacks (i.e., colluding managers increasing the scores), we use a minimum as voting function. Similarly, to blame a node p, a blame message is sent to p's managers. Finally, a node is expelled using the very same managers.

5.2 Direct verifications and cross-checking

In order to ensure *quantitative correctness*, the verification procedures check that the suspected node sent each proposal to f nodes (decrease fanout attack). To assess *causality*, the verification mechanism ensures that *(i)* a chunk proposed is served (partial serve attack), and that *(ii)* a chunk served to a node at a time t is proposed (partial propose attack) in its next propose phase, i.e., at the latest at time $t + T_g$ (decrease gossip period attack)

To ensure that received chunks are further proposed, we propose a direct *cross-checking* verification 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_{dcc}, to the set of f nodes to check whether they received or not a propose message from p_1 containing the chunks p_0 served to

p_1. The f witnesses reply with confirm responses confirming or infirming p_1's acknowledgment sent to p_0.

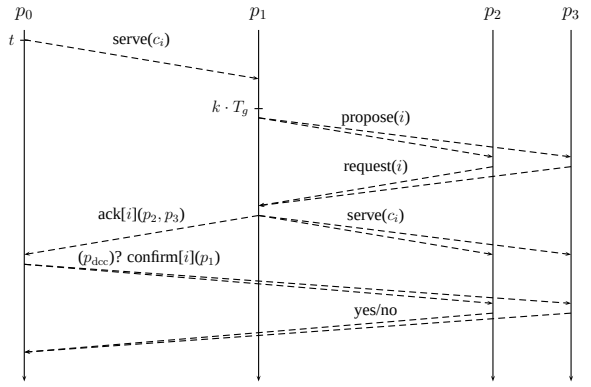


Figure 7. Cross-checking protocol.

Figure 7 depicts the message sequence composing a direct cross-checking procedure (with a fanout of 2 for the sake of readability). If anything wrong is detected, p_0 blames p_1 by the number of contradictory testimonies or by f if no acknowledgment is received. Direct cross-checking therefore ensures that every node forwards the chunks it receives to exactly f nodes in the next gossip period.

For the partial serve attack, it is straightforward for a node to detect that a chunk it has been proposed is effectively served. If not, the requester blames the proposer by $f/|\mathcal{R}|$ (where $\mathcal{R}$ is the set of requested chunks) for each chunk that has not been delivered. If the node did not serve any of the requested chunks, it is blamed by f which corresponds to the same blame as if the node did not propose those chunks.

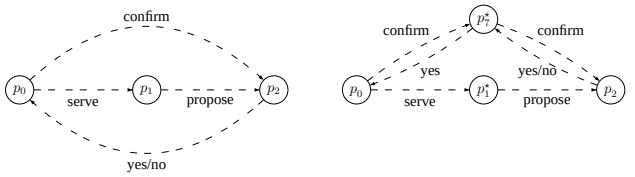
Since the verification messages (i.e., ack, confirm and confirm responses) for the direct cross-checking are small and in order to limit the subsequent overhead of LiFTinG, direct cross-checking is done exclusively with UDP.

The different attacks and corresponding blames are summarized in Table 1.

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 ($\lvert\mathcal{S}\rvert < \lvert\mathcal{R}\rvert$)	$f \cdot (\lvert\mathcal{R}\rvert - \lvert\mathcal{S}\rvert)/\lvert\mathcal{R}\rvert$ from the receiver

Table 1. Summary of attacks and associated blame values.

Fooling the direct cross-check ($\star$) Considering a set of colluding nodes, a node may freeride the protocol without being detected. Consider the situation depicted in Figure 8a, 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 8b. 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 statistical verifications presented in the next paragraph address this issue.



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

Figure 8. Direct cross-checking and attacks. Colluding nodes are denoted with a '$\star$'.

5.3 Local history auditing

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 paragraph, where freeriders collude to circumvent the verification procedures, highlights the need for statistical verification of a node's contacts, i.e., the nodes that exchanged messages with the audited node, to ensure that it does not collaborate only with a subset of the network, e.g., a set of colluding nodes that would cover each other up. The statistical verification of the history of the nodes, namely *local history auditing*, acts as a complement to the direct cross-checking. The history of a node that biased its partner selection contains a relatively large proportion of colluding nodes. If 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 entropy-based approach to detect the bias induced by freeriders on the history of nodes. If the history of the inspected node does not pass the entropic checks, the node is expelled from the system.

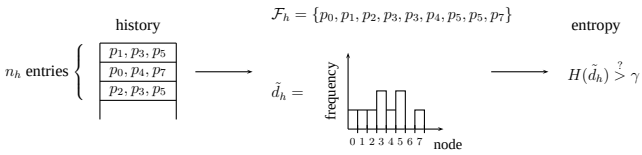


Figure 9. Local history auditing on proposals ($f = 3$).

Figure 9 gives a synthetic overview of local history auditing on proposals. The gossip protocol specifies that a node picks *uniformly at random*, a set of f partners every gossip period. When inspecting the local history of a node, the auditor computes the number of occurrences of each node in the set of proposals sent by p_1 during the last h seconds. Defining $\mathcal{F}_h$ as the multiset of nodes to whom p_1 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 $\mathcal{F}_h$. The empirical distribution $\tilde{d}$ is to be compared to the *uniform distribution*. The commonly used function to evaluate the similarity between two distributions is the Kullback-Leibler divergence. Measuring the uniformity of the distribution $\tilde{d}$ of p_1's partners boils down to computing its Shannon entropy:

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

The entropy is maximum when every node of the system appears at most once in $\mathcal{F}_h$ (assuming $n_h f < n$). In that case, it is equal to $\log_2(n_h f)$. Evaluating the *uniformity* of the partner selection is achieved by comparing the entropy of the partner distribution to a given threshold γ ($0 \leq \gamma \leq \log_2(n_h f)$). Since the peer selection service underlying the gossip protocol may not be perfect, the threshold must be tolerant to small deviation with respect to the uniform distribution to avoid *false positives* (i.e., honest nodes being blamed). Numerical values and practical applications are given in Section 6.

A local history audit 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.

So far we focused on inspecting the multi-set $\mathcal{F}_h$ of nodes to whom a node sent proposals. Due to the possibility of the man-in-the middle attack presented in Section 5.2 (using a colluding node to pass the direct verification), 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. When dealing with an honest node p_0, $\mathcal{F}'_h$ is composed of the nodes that sent chunks to p_0 – namely its *fanin*. Therefore, $\mathcal{F}'_h$ shares similar statistical properties with $\mathcal{F}_h$ and can thus be checked similarly, i.e., using an entropic check. On the other hand, when dealing with freeriders that implemented the man-in-the-middle attack, $\mathcal{F}'_h$ contains a large proportion of colluding nodes and thus the freeriders are detected. 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.

Local history auditing can be leveraged to check that a node respected the gossip period T_g specified by the protocol. Assuming a correct fanout (thanks to direct cross-checking), checking the gossip period boils down to counting the number of proposals in the local history.

To conclude, in addition to ensuring that statistical properties of the protocol are respected, local history auditing limits the possibility for freeriders to cover each other up in order to circumvent the verification procedures.

Local history auditing verifications are sporadically performed by the nodes using TCP connections. The reasons to use TCP are that *(i)* the overhead of establishing a connection is amortized since local history auditing happens sporadically and carries out a large amount of data, i.e., proportional to h, and *(ii)* local auditing is very sensitive to message losses as the potential blame is much larger than for direct verifications and can lead to expulsion from the system.

The different attacks and corresponding verification procedures are summarized in Table 2.

Attack	Type	Detection
fanout decrease (f)	quantitative	direct cross-check
partial propose ($\mathcal{P}$)	causality	direct cross-check
partial serve ($\mathcal{R}$)	quantitative	direct verification
decrease period (T_g)	quantitative	direct cross-check local auditing
bias partners selection ($\star$)	entropy	local auditing *a posteriori* cross-check

Table 2. Summary of attacks and associated verifications.

6. Analysis and simulation results

This section gives a theoretical performance analysis of the detection mechanism of LiFTinG. First we analyze its complexity, then we study probabilistically its detection property and the occurrence of false positives in the presence of colluding freeriders and message losses. The present analysis is backed up by Monte-Carlo simulations. Principal notations used in this section are summarized in Table 4 (page 10).

6.1 Communication costs

In this section, we evaluate the overhead caused by LiFTinG on the content dissemination protocol. To this end, we compute the maximum number of verification and blame messages sent by a node during one gossip period. The overhead of the verifications is summarized in Table 3. Note that we do not consider statistical verifications in this section as it does not imply a regular overhead but only sporadic message exchanges.

Direct verification Direct verifications do not require any exchange of verification messages as they consist only in comparing the number of chunks requested by the verifier to the number of chunks it really received. However, direct verification may lead to the emission of f blames (to M managers). The communication overhead caused by direct verifications is therefore $\mathcal{O}(M \cdot f)$ messages.

Direct cross-checking In order to check that the chunks it sent during the previous gossip period are further proposed, the verifier polls the f partners of its f partners with probability p_{dcc}. Similarly, a node is polled by $p_{\mathrm{dcc}} \cdot f^2$ nodes per gossip period on average and therefore sends $p_{\mathrm{dcc}} \cdot f^2$ replies. Finally, a node sends the list of its current partners to the f nodes (on average) that served chunks to it in the last gossip period. In addition, since a node inspects its f

partners, direct cross-checking may lead to the emission of a maximum of f blames (to M managers). The communication overhead caused by direct cross-checking is therefore $\mathcal{O}(p_{\text{dcc}} \cdot f^2 + p_{\text{dcc}} \cdot M \cdot f)$ messages. Setting p_{dcc} to $1/f$ the overhead is $\mathcal{O}(M + f)$.

The number of messages sent by LiFTinG is $\mathcal{O}(M \cdot f)$. This has to be compared to the number of messages sent by the three-phase gossip protocol itself, namely $f(2 + |\mathcal{R}|)$ – where $\mathcal{R}$ is the set of requested chunks, the two additional messages being the proposal and the request. The overhead of LiFTinG is even more negligible when taking into account the size of the chunks sent by a node, which is several orders of magnitude larger than the verification and blame messages. Finally, since $f \sim \ln(n)$, both the three-phase protocol and LiFTinG scale with the number of nodes.

direct verifications (messages)	0
direct verifications (blames)	$\mathcal{O}(M \cdot f)$ for the verifier
direct cross-check (messages)	$\mathcal{O}(p_{\text{dcc}} f^2)$ for the verifier $\mathcal{O}(p_{\text{dcc}} f)$ for the inspected node $\mathcal{O}(p_{\text{dcc}} f^2)$ for the each witness
direct cross-check (blames)	$\mathcal{O}(p_{\text{dcc}} \cdot M \cdot f)$ for the verifier

Table 3. Overhead of verifications.

6.2 Wrongful blames

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 misbehaviors. 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 [15] are traditionally used to address decision problems, i.e., 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. One may thus think that there is no need to compensate wrongful blames.

However, in the context of content distribution in the presence of freeriders, two problems arise when using relative score-based detection: *(i)* freeriders are able to decrease the probability to be 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 on the nodes is automatically compensated and the decision algorithm is based on absolute scores with a fixed threshold.

Considering message losses independently drawn from a Bernoulli distribution 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 during a given timespan. Periodically increasing all scores accordingly leads to an average score of 0 for honest nodes. This way, a fixed threshold 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 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., $p_{dcc} = 1$.

Direct verification This procedure checks, for each of the f partners to which a node sent a proposal, that the proposed and requested chunks are effectively received. 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, the node is therefore blamed by f ((a) in Equation 2). Otherwise it is blamed by the proportion of chunks lost ((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:

$$\begin{aligned} \tilde{b}_{\text{dv}} &= f \cdot \Big[\overbrace{p_r(1-p_r) \cdot f}^{\text{(a)}} + \overbrace{p_r^2 \cdot (1-p_r)\,|\mathcal{R}| \cdot \frac{f}{|\mathcal{R}|}}^{\text{(b)}} \Big] \\ \tilde{b}_{\text{dv}} &= p_r(1-p_r^2) \cdot f^2 \end{aligned} \tag{2}$$

Direct cross-checking This procedure checks that each chunk received is proposed to f nodes in the next gossip period. 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. If some chunks served by the verifier in the previous period are lost, then the verifier considers the f proposals sent by the inspected node as invalid, i.e., incomplete. The inspected node will therefore be blamed by f by this particular verifier ((a) in Equation 3). On the other hand, for each verifier and for each partner, the inspected node is blamed by 1 if the proposal, the confirmation or the answer to the confirmation is lost ((b) in Equation 3). The expected blame applied to an honest node (by the f verifiers), during one gossip period, due to message losses is therefore:

$$\begin{aligned} \tilde{b}_{\text{dcc}} &= f \cdot \Big[\overbrace{p_r^2(1-p_r^{|R|+1}) \cdot f}^{\text{(a)}} + \overbrace{f \cdot p_r^2 \cdot p_r^{|\mathcal{R}|+1}(1-p_r^3)}^{\text{(b)}} \Big] \\ \tilde{b}_{\text{dcc}} &= p_r^2(1-p_r^{|\mathcal{R}|+4}) \cdot f^2 \end{aligned} \tag{3}$$

A posteriori cross-checking This procedure asks the nodes that appear in the inspected node's history for confirmation which can cause wrongful blames. Effectively, if a proposal sent by the inspected node has not been received by the destination node, due to message losses, the latter will not acknowledge reception when asked. This leads again to wrongful blames. However, since the nodes are polled using TCP,

the polling message and the answer are not subject to message losses. On average, only $p_r \cdot n_h f$ proposals in the inspected node history are confirmed by the destination leading to an average blame of:

$$\tilde{b}_{\text{apcc}} = (1 - p_r) \cdot n_h f \quad (4)$$

Similarly to direct verification, the wrongful blames applied by the local auditing must be compensated. However, this should be done only sporadically, i.e., only when a node is effectively audited, since these verifications are not triggered at each gossip period.

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

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

Following the same line of reasoning, a closed form expression for the standard deviation $\sigma(b)$ of b can be derived [8].

Figure 10 depicts the distribution of scores after one gossip period in a simulated network of $10,000$ honest nodes in steady state (where both direct verifications and direct cross-checking are performed with $p_{dcc} = 1$). The message loss rate p_l has been 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 (5). We observe that, as expected, the average score (dotted line) is close to zero (< 0.01) which means that the wrongful blames have been successfuly compensated. The experimental standard deviation is $\sigma(b) = 25.6$.

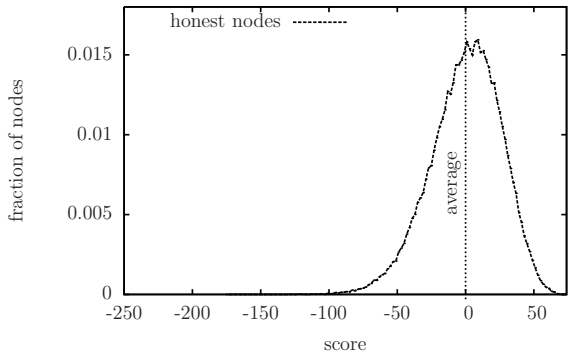


Figure 10. Impact of message losses.

6.3 Detection and false positives

A node can be expelled from the system either when its score drops beyond a fixed threshold (η) or upon a local auditing procedure. 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) in both situations.

6.3.1 Score-based detection

As mentioned above, the score-based detection mechanism uses a fixed threshold η to which the scores of the nodes are compared. To this aim, the score of each node is *adjusted* (to compensate wrongful blames) and *normalized* by the number of gossip periods r the node spent in the system. At the t-th gossip period, the normalized score of a node writes:

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

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 i.i.d. (independent and identically distributed) we get $\mathbb{E}[s] = 0$ and $\sigma(s) = \sigma(b)/\sqrt{n_T}$. Using Bienaymé-Tchebychev's inequality we derive an upper bound for the probability of false positive:

$$\beta = P\left(s < \eta\right) \leq P\left(|s| > -\eta\right) \leq \frac{\sigma(b)^2}{r \cdot \eta^2}$$

The probability α to catch a freerider depends on its *degree of freeriding* that characterizes its deviation to the protocol. Formally, we define the degree of freeriding as a 3-uple $\Delta = (\delta_1, \delta_2, \delta_3)$, $0 \leq \delta_1, \delta_2, \delta_3 \leq 1$, so that a freerider contacts only $\delta_1 \cdot f$ nodes per gossip period, proposes the chunks received from a proportion δ_2 of the nodes that served it in the previous gossip period[1], and serves $\delta_3 \cdot |\mathcal{R}|$ chunks to each requesting node. The gain in terms of the upload bandwidth saved is therefore $1 - (1 - \delta_1)(1 - \delta_2)(1 - \delta_3)$.

Following the same line of reasoning as in the previous section, we can derive a closed form expression for the expected blame applied to a freerider as a function of Δ:

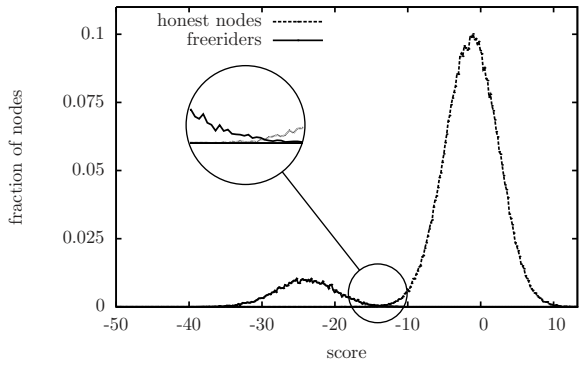
$$\begin{aligned}\tilde{b}'(\Delta) \;=\; & (1-\delta_1)\cdot p_r\left(1 - p_r^2(1-\delta_3)\right)\cdot f^2 + \delta_2 \cdot f^2 + \\ & (1-\delta_2)\cdot p_r^2 \cdot \Big[p_r^{|\mathcal{R}|+1}(1 - p_r^3(1-\delta_1)) + \\ & \qquad (1 - p_r^{|\mathcal{R}|+1})\Big]\cdot f^2\end{aligned}$$

Similarly to $\sigma(b)$, a closed form expression for the standard deviation $\sigma(b'(\Delta))$ of $b'(\Delta)$ can be obtained [8]. Similarly to the probability of false positivesβ, the probability of detection α can be lower bounded:

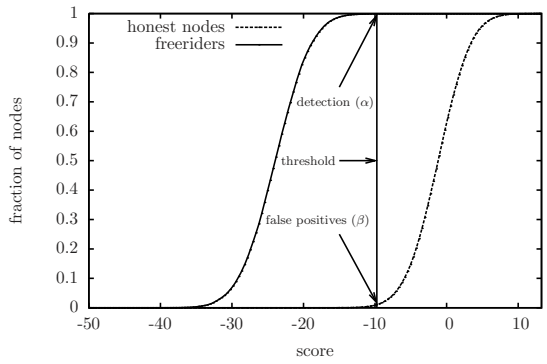
$$\alpha \geq 1 - \frac{\sigma(b'(\Delta))^2}{r \cdot (\tilde{b'}(\Delta) - \eta)^2}$$

Note that the performance of LiFTinG increases over time. Effectively, as the detection threshold is fixed and the standard deviations of the score distributions tend to zero when the time spend in the system increases, the probability of detection α increases to one and the probability of false positive β decreases to zero.

[1] When a node removes some chunks from its proposals, it is blamed by the nodes that served them. A freerider has therefore interest in removing chunks from the least number of sources.



(a) probability density function (pdf)



(b) cumulative distribution function (cdf)

Figure 11. Distribution of normalized scores in the presence of freeriders ($\Delta = (0.1, 0.1, 0.1)$).

Figure 11 depicts the distribution of normalized scores in the presence of $1,000$ freeriders of degree $\Delta = (0.1, 0.1, 0.1)$ in a $10,000$-node system after $r = 50$ gossip periods. We plot separately the distribution of scores among honest nodes and freeriders. As expected, the probability density function (Figure 11a) 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. Figure 11b depicts the cumulative distribution function of scores and illustrates the notion of detection and false positives for a given value of the detection threshold (i.e., $\eta = -9.75$).

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 the same probability ($\delta_1 = \delta_2 = \delta_3 = \delta$) and we observe the proportion of freeriders detected by LiFTinG for several values of δ. Figure 12 plots α and β as functions of δ. For instance, we observe that for a node freeriding by 5%, the probability to be detected by LiFTinG is 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 [22], 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 12).

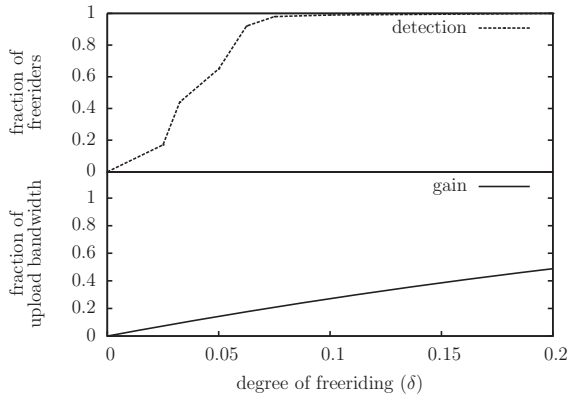


Figure 12. Proportion of freeriders detected by LiFTinG.

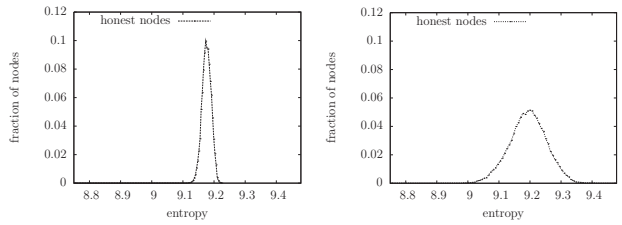
6.3.2 Entropy-based detection

For the sake of fairness and in order to prevent colluding nodes from covering each other up, LiFTinG includes a local audit assessing the statistical validity of the partner selection. To this end, the entropy of the distribution of the inspected node's former partners is compared to a threshold γ, which is a parameter of the system. The distribution of the entropy H of an honest node's history depends on the peer sampling algorithm used and can be estimated by simulations.

Figure 13a depicts the distribution of entropy H for a history of $n_h f = 600$ partners ($n_h = 50$ and $f = 12$) in 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 $\log_2(n_h f) = 9.23$. Similarly, the entropy of the multi-set $\mathcal{F}'_h$, of nodes that selected the inspected node as partner, i.e., its fanin, is depicted in Figure 13b. The observed entropy ranges from 8.98 to 9.34. Note that the size of $\mathcal{F}'_h$ can be greater than $n_h f$ (but is $n_h f$ on average) and therefore the bound $\log_2(n_h f)$ does not apply to the entropy of fanin.

The simulation results show that the probability of wrongfully expelling the inspected node during local auditing is negligible when the threshold γ is set to 8.95. This threshold is used for both fanout and fanin entropic check.

We now determine analytically 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'. A first requirement to be able to detect colluding nodes is that the number of proposals in a node's history must be greater than the number of colluding freeriders. Otherwise, by proposing chunks only to other freeriders in a round-robin manner, a node may still be able to achieve a maximized entropy. We therefore set h so that $n_h f \gg m'$. 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



(a) entropy of fanout (b) entropy of fanin

Figure 13. Distribution of the entropy H of the nodes' histories using a full membership-based partner selection.

$1 - p_m$. We seek the maximum value $p_m^\star$ for p_m, function of γ and m'. Defining the probability law of the partner selection among honest nodes (resp. colluding nodes) by X (resp. by Y), the entropy of its fanout is expressed as follows:

$$\begin{aligned} H(\mathcal{F}_h) &= p_m \log_2 p_m + (1-p_m)\log_2(1-p_m) \\ &\quad + p_m H(X) + (1-p_m)H(Y) \ , \end{aligned}$$

since X and Y are independent. This quantity is maximized when X and Y are the uniform distribution. Therefore, 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. Therefore, given a threshold γ and a maximum number of colluding nodes m', we have:

$$\gamma = -p_m^\star \log_2 \left(\frac{p_m^\star}{m'}\right) - (1-p_m^\star)\log_2\left(\frac{1-p_m^\star}{n_h f - m'}\right) \qquad (7)$$

where $p_m^\star$ is the maximum value for p_m that a freerider can use without being detected. Inverting numerically Formula (7), we deduce that for $\gamma = 8.95$ a freerider colluding with 25 other nodes can serve its colluding partners 21% of the time, without being detected. In this setting, a freerider can therefore further decrease its contribution by 21%.

Notations	Descriptions
n,m	number of nodes / freeriders
$\lvert\mathcal{R}\rvert$	number of chunks requested
f	fanout
n_h	size of history
$\mathcal{F}_h, \mathcal{F}'_h$	multi-set of fanout / fanin in history
p_{dcc}	probability to trigger direct cross-checking
p_l	probability of message loss ($p_r = 1 - p_l$)
$\bar{b}$	average value of wrongful blames
$\sigma(b)$	standard deviation of wrongful blames
r	number of gossip periods spent in the system
s	normalized score
Δ	degree of freeriding (3-uple)
$\bar{b}(\Delta)$	average value of blames (freeriders)
$\sigma(b'(\delta))$	standard deviation of blames (freeriders)
η	detection threshold (blame-based detection)
α	probability of detection (blame-based detection)
β	probability of false positive (blame-based detection)
γ	detection threshold (entropy-based detection)

Table 4. Summary of principal notations.

7. Evaluation and experimental results

We now evaluate LiFTinG on top of the gossip-based streaming protocol described in [6], over the PlanetLab testbed.

7.1 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.

7.2 Practical cost

Table 5 gives the bandwidth overhead of the direct verifications of LiFTinG for three values of p_{dcc}. Note that the overhead is not null when $p_{\mathrm{dcc}} = 0$ since ack messages are always sent. Yet, we observe that the overhead is negligible when $p_{\mathrm{dcc}} = 0$ (i.e., when the system is healthy) and remains reasonable when $p_{\mathrm{dcc}} = 1$ (i.e., when the systems needs to be purged from freeriders).

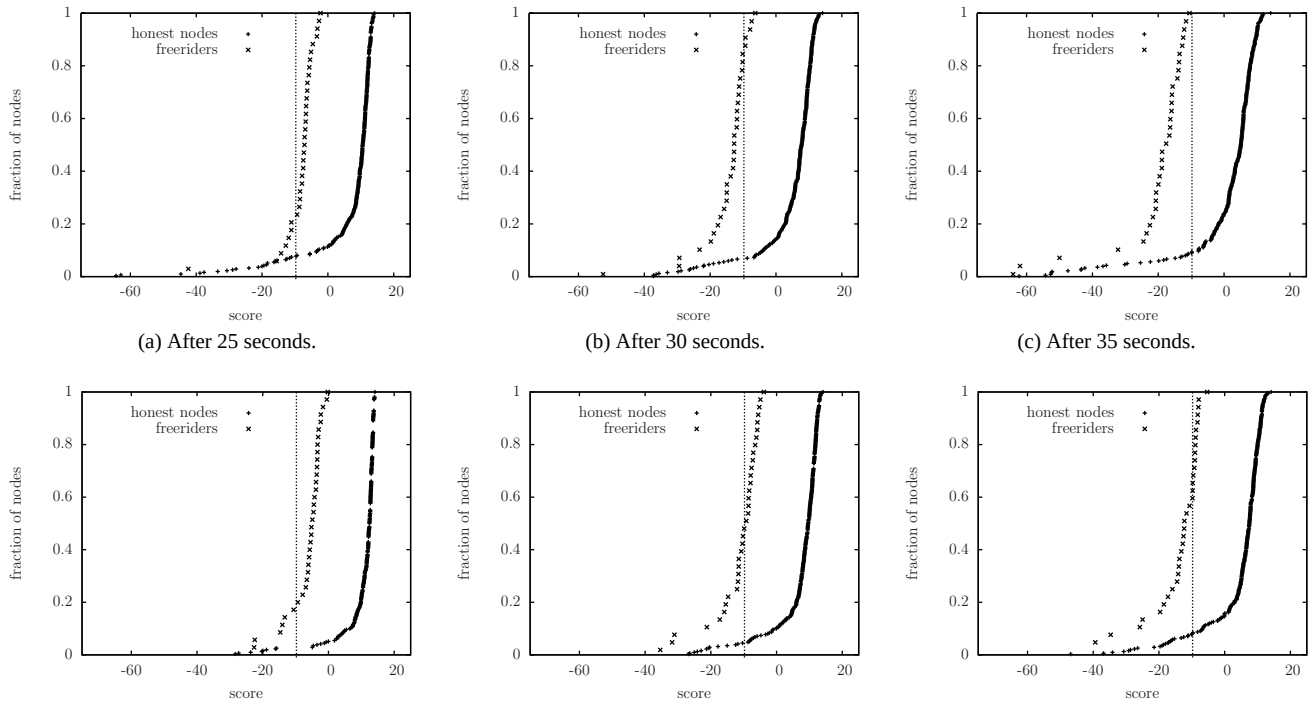
	cross-checking and blaming overhead		
$\mathbf{p_{dcc}}$	0	0.5	1
674 kbps stream	1.07%	4.53%	8.01%
1082 kbps stream	0.69%	3.51%	5.04%
2036 kbps stream	0.38%	1.69%	2.76%

Table 5. Practical overhead

7.3 Experimental results

We have executed LiFTinG with $p_{dcc} = 1$ and $p_{dcc} = 0.5$. Figure 7 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 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 14b ($p_{dcc} = 1$, 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_{dcc} 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 verification blames freeriders without the need for any cross-check.



(a) After 25 seconds. (b) After 30 seconds. (c) After 35 seconds.

(d) After 25 seconds. (e) After 30 seconds. (f) After 35 seconds.

Figure 14. Cumulative distribution functions of scores with $p_{dcc} = 1$ (above) and $p_{dcc} = 0.5$ (below).

This explains why the detection after only 35 seconds with $p_{dcc} = 0.5$ (Figure 14f) is comparable with the detection after 30 seconds with $p_{dcc} = 1$ (Figure 14b).

As stated in the analysis, we observe that the gap between the two cdfs 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.

8. Related Work

The novel approach proposed in this paper aims at preventing freeriding in gossip-based dissemination protocols via coercion. It is related to the following recent 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 are a number of attacks, mainly targeting the opportunistic unchoking mechanism, allowing freeriders to download contents with no or a very small contribution [23, 24].

BAR Gossip [21] and its improvement [22] is a gossip-based streaming application that fights against freeriding using verifications on partner selection and chunk exchanges. BAR Gossip 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 based on a TfT mechanism. 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. BAR Gossip 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 [11] that BAR Gossip presents scalability issues, not to mention the overhead of cryptography.

PeerReview [9] deals with malicious nodes following an accountability approach. Peers maintain verifiable signed logs of their actions that can be audited and checked using a simulator of the application, i.e., 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 case of malicious participants was considered in the context of generic gossip protocols, i.e., consisting of state exchanges and updates [12]. This system relies on cryptography for signing messages between peers and do not con-

sider malicious behaviors that stem from the partner selection, i.e., biasing the random choices. In addition, they do not tackle the problem of collusion.

The two approaches that relate the most to LiFTinG are the distributed auditing protocol proposed in [11] and the passive monitoring protocol proposed in [14]. The first protocol targets live streaming applications. 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, i.e., 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.

The originality of our work lies in exploiting the inherent randomness of gossip-based protocols, information cross-checking and statistical verifications to fight against freeriders, even in presence of colluding nodes. The strength of LiFTinG is to achieve efficient freeriders-tracking with a slight overhead in challenging environment (i.e., facing message losses and asynchrony).

9. Conclusion

We presented LiFTinG, a protocol for tracking freeriders in gossip-based asymmetric data dissemination systems. Beyond the fact that LiFTinG deals with the inherent randomness of the protocol, LiFTinG precisely relies on this randomness to robustify its verification mechanisms against colluding freeriders with a very slight 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 plan to integrate LiFTinG into an existing symmetric content dissemination system to secure its opportunistic unchoking mechanism.

References

[1] E. Adar and B. Huberman. Free riding on Gnutella. *First Monday*, 5(10), October 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] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg. Epidemic Live Streaming: Optimal Performance Trade-offs. In *SIGMETRICS*, 2008.

[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] D. Frey, R. Guerraoui, A.-M. Kermarrec, M. Monod, and V. Quéma. Stretching Gossip with Live Streaming. In *DSN*, 2009.

[7] J. Gerard, H. Cai, and J. Wang. Alliatrust: A Trustable Reputation Management Scheme for Unstructured P2P Systems. In *GPC*, 2006.

[8] R. Guerraoui, K. Huguenin, A.-M. Kermarrec, and M. Monod. LiFT: Lightweight Freerider-Tracking Protocol. Research Report RR-6913, INRIA, 2009.

[9] A. Haeberlen, P. Kouznetsov, and P. Druschel. PeerReview: Practical Accountability for Distributed Systems. In *SOSP*, 2007.

[10] G. Hardin. The Tragedy of the Commons. *Science*, 162:1243–1248, 1968.

[11] M. Haridasan, I. Jansch-Porto, and R. Van Renesse. Enforcing Fairness in a Live-Streaming System. In *MMCN*, 2008.

[12] M. Jelasity, A. Montresor, and O. Babaoglu. Detection and Removal of Malicious Peers in Gossip-Based Protocols. In *FuDiCo*, 2004.

[13] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based Peer Sampling. *TOCS*, 25(3): 1–36, 2007.

[14] M. Karakaya, I. Körpeoğlu, and O. Ulusoy. Counteracting Free-riding in Peer-to-Peer Networks. *Computer Networks*, 52(3):675–694, 2008.

[15] S. Kay. *Fundamentals of Statistical Signal Processing: Estimation Theory*. Prentice Hall, 1993.

[16] A.-M. Kermarrec, L. Massoulié, and A. Ganesh. Probabilistic Reliable Dissemination in Large-Scale Systems. *TPDS*, 14 (3):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, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR Gossip. In *OSDI*, 2006.

[22] H. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robinson, L. Alvisi, and M. Dahlin. FlightPath: Obedience v.s. Choice in Cooperative Services. In *OSDI*, 2008.

[23] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free Riding in BitTorrent is Cheap. In *HotNets*, 2006.

[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.