LiFTinG: Lightweight Freerider-Tracking Protocol in Gossip

Rachid Guerraoui, Kévin Huguenin, Anne-Marie Kermarrec, Maxime Maxime Monod, Swagatika Prusty

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

LiFTinG: Lightweight protocol for Freerider-Tracking in Gossip

Rachid Guerraoui — Kévin Huguenin — Anne-Marie Kermarrec — Maxime Monod — Swagatika Prusty

N° 6913

December 2009

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--6913--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

LiFTinG: Lightweight protocol for Freerider-Tracking in Gossip

Rachid Guerraoui* , Kévin Huguenin , Anne-Marie Kermarrec , Maxime Monod* , Swagatika Prusty†

Thème : Réseaux, systèmes et services, calcul distribué
Équipes-Projets Asap

Rapport de recherche n° 6913 — December 2009 — 21 pages

Abstract: This report 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.

Key-words: Peer-to-peer systems, Gossip-based protocols, Freeriders, Accountability

* Laboratoire de programmation distribuée (LPD), École Polytechnique Fédérale de Lausanne (EPFL)
† IIT Guwahati

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

LiFTinG : un protocole léger de détection de pair égoïste

Résumé :

Mots-clés :

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] (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 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. In addition, freeriders may collude, i.e., collaborate to decrease their individual and common contribution to the system and cover each other up to circumvent detection mechanisms.

By using the Tit-for-Tat (TfT) incentives (inspired from file-sharing systems [4]), TfT-based 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 that ensure that nodes which do not contribute their fair share do not benefit anymore 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 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 difficulty at first glance, happens to help tracking freeriders. Effectively, LiFTinG exploits the very fact that nodes pick neighbors at random to prevent 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. This overhead can be dynamically adjusted and potentially reduced to zero when the system is healthy. In addition, LiFTinG is fully decentralized as nodes are in charge of verifying each others' actions and monitoring each others' behavior. 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, very low.

To evaluate LiFTinG, we give analytical results backed up with simulations, providing means to set up 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. 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).

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 was smaller than required. However, this is due to poor capabilities, as opposed to freeriders that deliberately decrease their contribution.

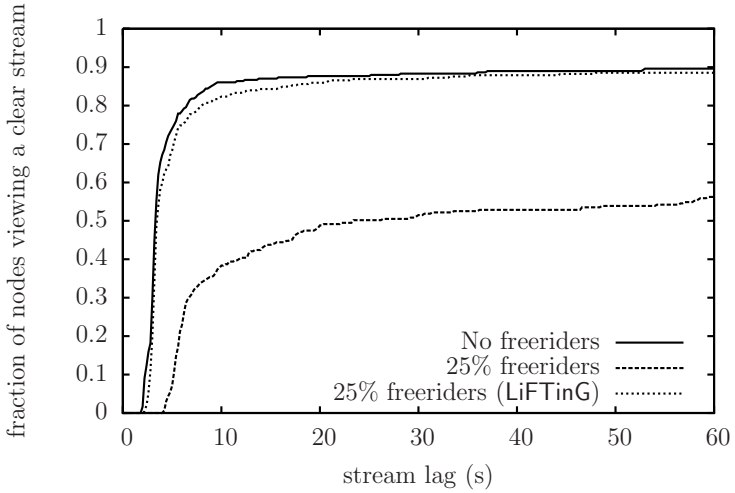


Figure 1: System efficiency in the presence of freeriders.

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, but still request content.

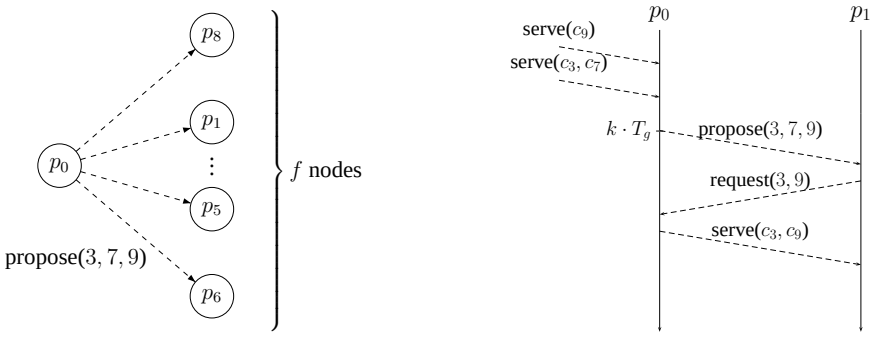
The rest of the paper is organized as follows. Section 2 describes our illustrative gossip protocol and Section 3 lists and classifies the opportunities for nodes to freeride. Section 4 presents LiFTinG and Section 5 formally analyzes its performance backed up by extensive simulations. Section 6 reports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

2 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.

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.



(a) Gossip dissemination (b) Three phases

Figure 2: Three-phase generic gossip.

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 proposed verifications of LiFTinG. Freeriders allow themselves to deviate 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 cover 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* [11]. We denote by m the number of freeriders.

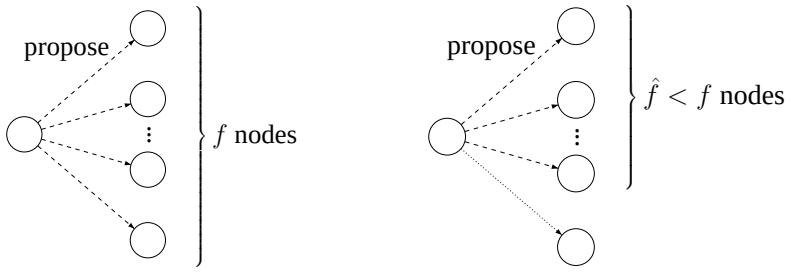
Freeriders may deviate from the gossip protocol in three ways: *(i)* bias the partner selection, *(ii)* drop messages they are supposed to send, or *(iii)* modify the content of the messages they send. In the sequel, we exhaustively list all possible attacks in each phase of the protocol, discuss their 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 '$\star$'. The analysis on the possible freeriding attacks to the three-phase protocol is at the core of LiFTinG.

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, as illustrated in Figure 3, 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.

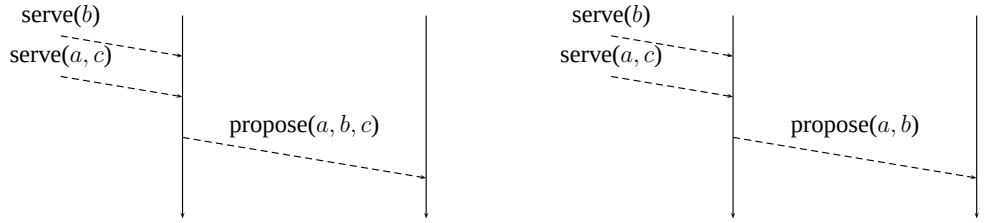
(ii) **Invalid proposal** A proposal is valid if it contains every chunk received in the last gossip period. Proposing only a subset of the chunks received in the last period, as illustrated in Figure 4, obviously decreases the number of requested chunks. However, a freerider has no interest in proposing chunks it does not have since, contrarily to TfT-based protocols, uploading chunks to a node does not imply



(a) Honest node (b) Freerider

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

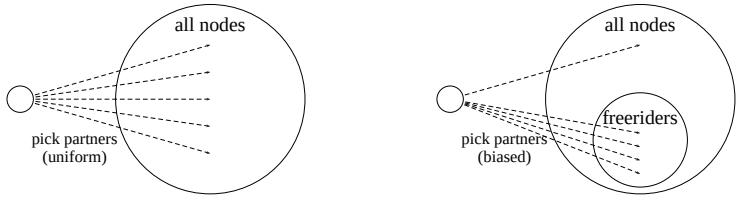
that the latter sends chunks in return. In other words, proposing more (and possibly fake) chunks does not increase the benefit of a node and does thus not need to be considered.



(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.



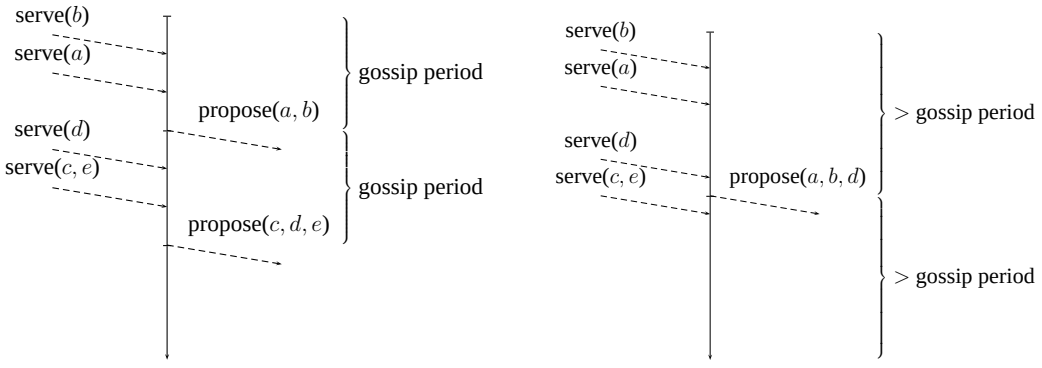
(a) Honest node (b) Colluding freeriders

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 advertising more, but "older", chunks per proposal 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.

3.2 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.



(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).

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 of what it sends.

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.

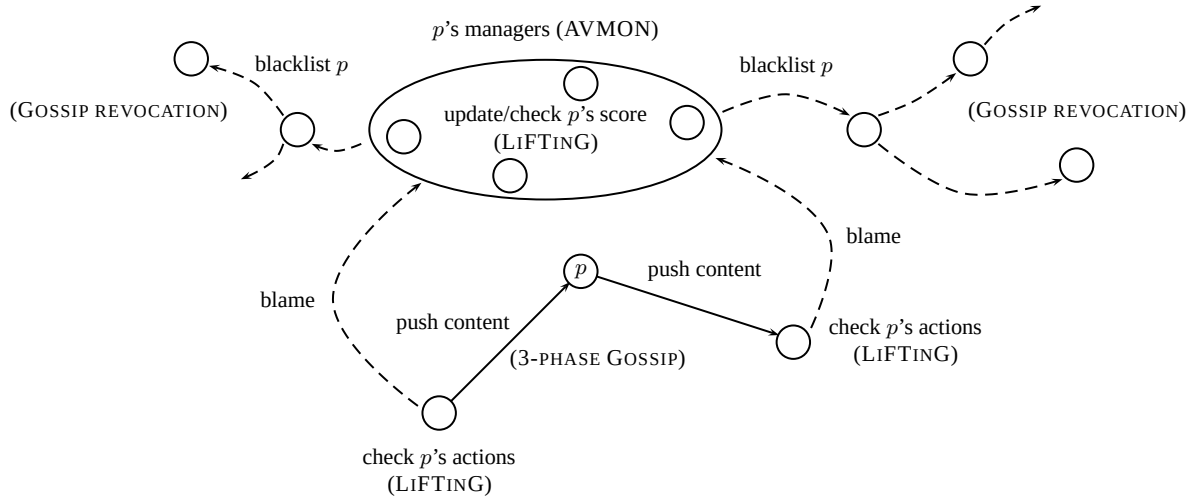


Figure 7: Overview of LiFTinG

We present the blaming architecture in Section 4.1 and present direct verifications in details 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.

We analyze the message complexity of LiFTinG in Section 4.5. 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 ($\lvert\mathcal{S}\rvert < \lvert\mathcal{R}\rvert$)	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 the score of a node p drops beyond a *fixed* threshold (the design choice of using a fixed threshold is explained in details in Section 5.1), the managers spread – through gossip – a revocation message against p making the nodes of the system progressively remove p from their membership. A representation of blame messages sent to p's managers and revocation messages gossiped from those managers to other participants in case p's score goes below a threshold is synthesized in Figure 7.

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 from 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 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 at ensuring 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 infirming p_1's acknowledgment sent to p_0.

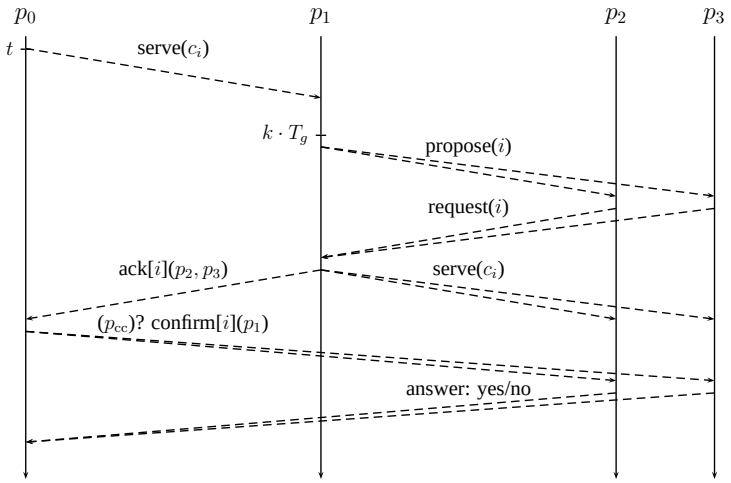


Figure 8: Cross-checking protocol.

Figure 8 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.

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 each requester

Table 2: Summary of attacks and associated blame values.

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 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 ($0 \leq \delta \leq 1$). 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 $f_{in} \cdot (f - \hat{f}) = f_{in} \cdot \delta \cdot f \simeq \delta f^2$.

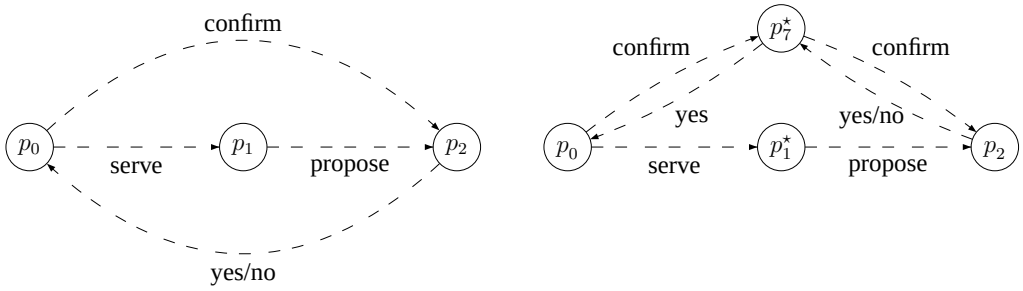
(ii) Partial propose (direct cross-check): If p_f proposes only $(1 - \delta) \cdot c$ chunks to f nodes, it is blamed by f by each of the node that provided at least one of the missing chunks. A freerider has therefore interest in removing from its proposal chunks originating from the smallest subset 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 chunk by each of the f requesting nodes. This again results in a blame value of $f \cdot (f/|\mathcal{R}|) \cdot \delta \cdot |\mathcal{R}| = \delta \cdot f^2$.

The blame values emitted by the different direct verifications are therefore homogeneous and comparable on average since all misbehaviors lead to the same amount of blame for a given degree of freeriding δ. 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 9a, 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 9b. 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

Figure 9: 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 10.

When inspecting the local history of a node, the verifier computes the number of occurrences of each node in the set of proposals sent by p 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 $\mathcal{F}_h$. Assessing

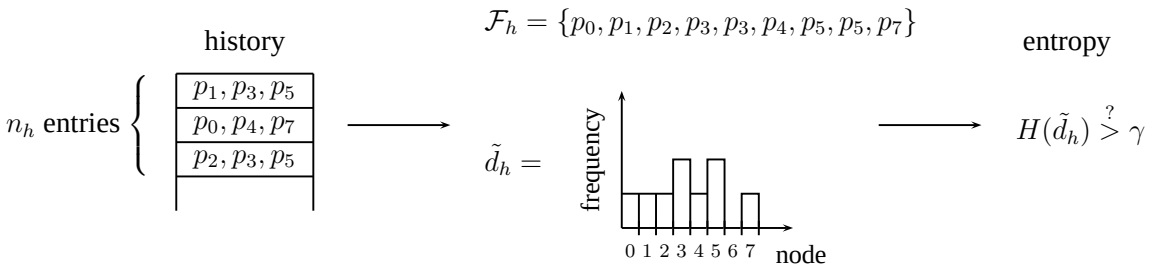


Figure 10: Entropic check on proposals ($f = 3$).

the uniformity of the distribution $\tilde{d}$ of p_1's history is achieved by comparing its Shannon entropy to a threshold γ ($0 \leq \gamma \leq \log_2(n_h f)$).

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

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.

Because of the 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.

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 it can lead to expulsion from the system.

4.5 Communication costs

In this section, we evaluate the overhead incurred by LiFTinG. To this end, we compute the maximum number of verification and blame messages sent by a node during one gossip period. The overheads of the verifications are 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 check Verifying that what was requested is actually served does not require any exchange of verification messages as direct check consists only in comparing the number of chunks requested by the verifier to the number of chunks it really received. However, direct check may lead to the emission of f blames, i.e., a blame for each sender (to M managers). The communication overhead caused by direct checking 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_{cc}, i.e., sending confirm messages. Similarly, a node is polled by $p_{\text{cc}} \cdot f^2$ nodes per gossip period on average and therefore sends $p_{\text{cc}} \cdot f^2$ answers to confirm messages. Finally, a node sends the list of its current partners (i.e., ack messages) to the f nodes (on average) that served chunks to it in the last gossip period, i.e., sending of

the ack messages. 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 $\mathcal{O}(p_{\text{cc}} \cdot f^2 + p_{\text{cc}} \cdot M \cdot f + f)$ messages.

The number of messages sent by LiFTinG is $\mathcal{O}(M \cdot f + f^2)$. This has to be compared to the number of messages sent by the three-phase gossip protocol itself, namely $f(2 + |\mathcal{S}|)$, where $\mathcal{S}$ is the set of served chunks, the two additional messages being the proposal and the request. The overhead of LiFTinG is 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. Note that M is a system parameter defining the number of managers of a node and does not depend on the size of the system. 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)	$\mathcal{O}(p_{\text{cc}} f^2)$ for the verifier (confirm messages) $\mathcal{O}(p_{\text{cc}} f)$ for the inspected node (ack messages) $\mathcal{O}(p_{\text{cc}} f^2)$ for each witness (answer messages)
direct cross-check (blames)	$\mathcal{O}(p_{\text{cc}} \cdot M \cdot f)$ for the verifier

Table 3: Overhead of verifications.

5 Parametrizing LiFTinG

This section provides a methodology to set LiFTinG's parameters. To 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.

Principal notations used in this section are summarized in Table 4 (page 18).

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 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 are traditionally 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 and detection thus consists in comparing the nodes' compensated scores to a fixed threshold η. In short, when the compensated score of a node goes 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 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 by direct verifications during a given timespan. Periodically increasing 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., $p_{\text{cc}} = 1$.

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:

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

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).

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

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:

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

Similarly to direct verifications, 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 by direct verifications due to message losses:

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

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

Figure 11 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 $p_{\text{cc}} = 1$). 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 successfully compensated. The experimental standard deviation is $\sigma(b) = 25.6$.

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

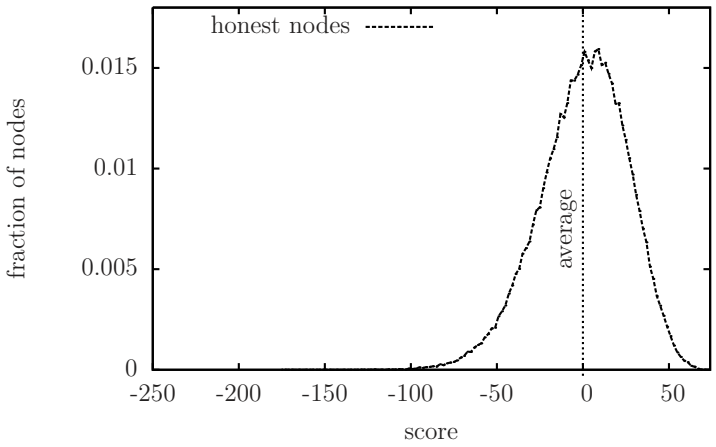


Figure 11: Impact of message losses.

of detection α) and the proportion of honest nodes wrongfully expelled from the system (probability of false positives β) in both situations.

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:

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

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 $(1-\delta_1)\cdot f$ 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. The gain in terms of the upload bandwidth saved is therefore $1-(1-\delta_1)(1-\delta_2)(1-\delta_3)$.

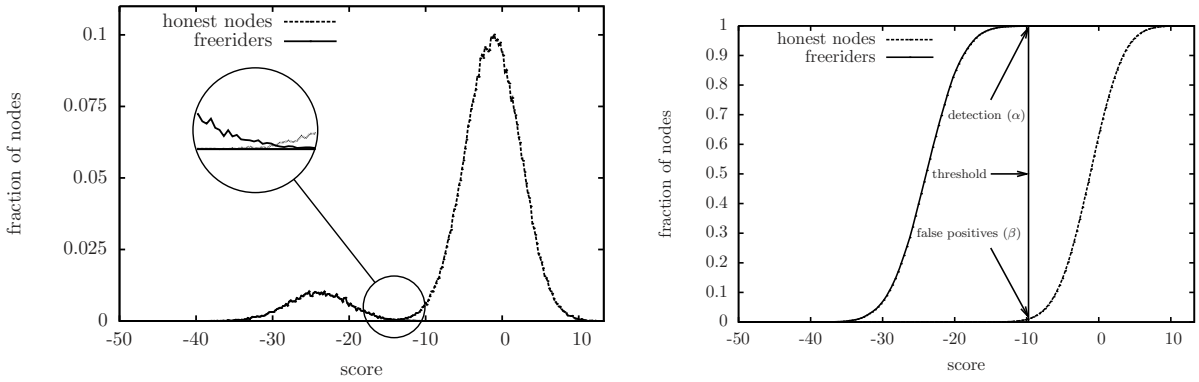
Following the same line of reasoning as in the previous section, we can derive a closed form expression for the expected blame applied to a freerider as a function of Δ:

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

Similarly to $\sigma(b)$, a closed form expression for the standard deviation $\sigma(b'(\Delta))$ of $b'(\Delta)$ can be obtained. Similarly to the probability of false positives β, 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 under the assumption that losses are independently drawn from a Bernoulli distribution the performance of LiFTinG increases over time. Effectively, as the detection threshold is fixed and the



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

Figure 12: Distribution of normalized scores in the presence of freeriders ($\Delta = (0.1, 0.1, 0.1)$).

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.

Figure 12 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 12a) 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 12b 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 13 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 [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 13).

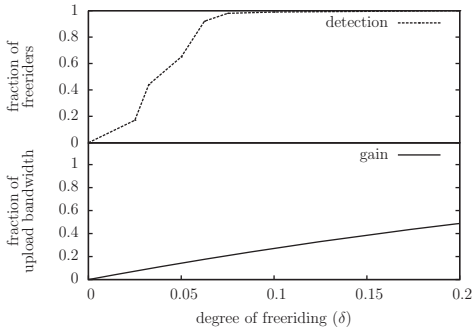


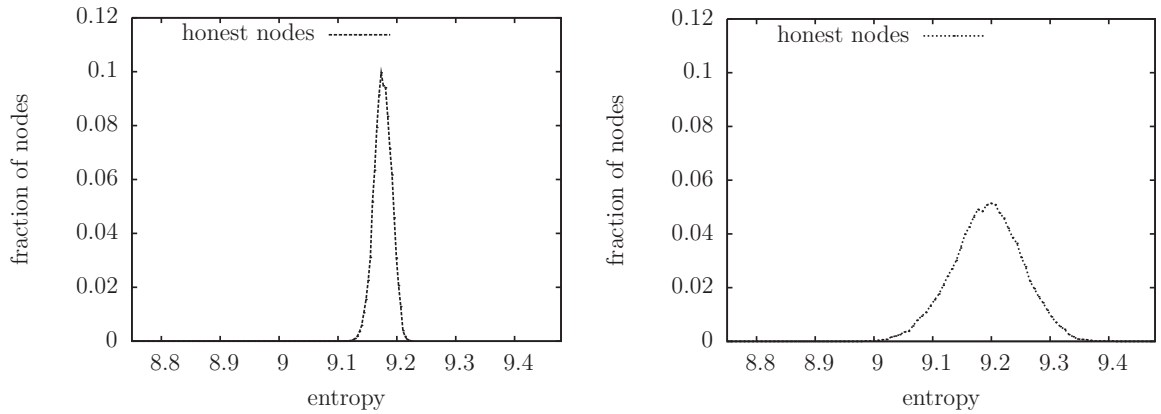
Figure 13: Proportion of freeriders detected by LiFTinG.

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, which is a parameter of the system. The distribution of the entropy of honest nodes' histories depends

on the peer sampling algorithm used and can be estimated by simulations. Figure 14a 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 $\log_2(n_h f) = 9.23$. Similarly, the entropy of the fanin multi-set $\mathcal{F}'_h$, e.g., nodes that selected the inspected node as partner, is depicted in Figure 14b. 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 presented results show that the probability of wrongfully expelling an 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.



(a) entropy of fanout

(b) entropy of fanin

Figure 14: Distribution of the entropy H of the nodes' histories using a full membership-based partner selection.

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'. 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 $1 - p_m$. 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 colluding nodes (resp. honest nodes) by X (resp. by Y), the entropy of its fanout writes:

$$\begin{aligned}
H(\mathcal{F}_h) &= -\sum p(n)\log_2(p(n)) \\
&= -\sum_{n\in X} p(n)\log_2(p(n)) - \sum_{n\in Y} p(n)\log_2(p(n)) \\
&= -\sum_{n\in X} p_m p_X(n)\log_2(p_m p_X(n)) - \sum_{n\in Y}(1-p_m)p_Y(n)\log_2((1-p_m)p_Y(n)) \\
&= -p_m \sum_{n\in X}(p_X(n)\log_2(p_X(n)) + p_X(n)\log_2(p_m)) \\
&\quad -(1-p_m)\sum_{n\in Y}(p_Y(n)\log_2(p_Y(n)) + p_Y(n)\log_2(1-p_m)) \\
&= -p_m\left(\log_2(p_m)\underbrace{\sum_{n\in X} p_X(n)}_{=1} + \underbrace{\sum_{n\in X}(\log_2(p_X(n))\cdot p_X(n))}_{=-H(X)}\right) \\
&\quad -(1-p_m)\left(\log_2(1-p_m)\underbrace{\sum_{n\in Y} p_y(n)}_{=1} + \underbrace{\sum_{n\in Y}(\log_2(p_Y(n))\cdot p_Y(n))}_{-H(Y)}\right) \\
&= -p_m\log_2 p_m - (1-p_m)\log_2(1-p_m) + p_m H(X) + (1-p_m)H(Y) \ ,
\end{aligned}$$

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 can determine the maximum colluding factor $p_m^\star$ a freerider can use without being detected. We first derive an expression of $H(X)$ knowing the number of occurrences of each m' colluding freeriders in the fraction of the history composed of freeriders ($p_m \cdot n_h \cdot f$), that is $\frac{p_m \cdot n_h \cdot f}{m'}$. The probability of a given freerider to appear at a given position in this fraction of the history is therefore $\frac{p_m \cdot n_h \cdot f}{m'}/(p_m \cdot n_h \cdot f) = 1/m'$, and thus we have $H(X) = -\sum_{m'} \frac{1}{m'}\log_2\left(\frac{1}{m'}\right) = -\log_2(m')$ and

$$H(Y) = -\sum_{n_h\cdot f(1-p_m)} \frac{1}{n_h\cdot f(1-p_m)}\log_2\left(\frac{1}{n_h\cdot f(1-p_m)}\right) = -\log_2\left(\frac{1}{n_h\cdot f(1-p_m)}\right) \ .$$

Finally, we get

$$\begin{aligned}
\gamma &= -p_m^\star\log_2 p_m^\star - (1-p_m^\star)\log_2(1-p_m^\star) - p_m^\star\log_2\left(\frac{1}{m'}\right) + (1-p_m^\star)(-\log_2\left(\frac{1}{n_h\cdot f(1-p_m^\star)}\right) \\
&= -p_m^\star\log_2\left(\frac{p_m^\star}{m'}\right) - (1-p_m^\star)\log_2\left(\frac{1}{n_h\cdot f}\right) \qquad (7)
\end{aligned}$$

where $p_m^\star$ is the maximum value for the favoring factor 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 15% of the time, without being detected. In this setting, a freerider can therefore decrease its contribution by 15%.

6 Evaluation and experimental results

We now evaluate LiFTinG on top of the gossip-based streaming protocol described in [7], over the PlanetLab testbed.

Notations	Descriptions
n, m	number of nodes / freeriders
$\|\mathcal{R}\|$, $\|\mathcal{S}\|$	number of chunks requested, resp. served
f	fanout
n_h	size of history
$\mathcal{F}_h, \mathcal{F}'_h$	multi-set of fanout / fanin in history
p_{cc}	probability to trigger cross-checking
p_l	probability of message loss ($p_r = 1 - p_l$)
$\tilde{b}$	average value of wrongful blames
$\sigma(b)$	standard deviation of wrongful blames
r	number of gossip periods spent in the system
s	normalized score
$\Delta = (\delta_1, \delta_2, \delta_3)$	degree of freeriding (3-uple)
$\delta = \delta_1 = \delta_2 = \delta_3$	degree of freeriding
$\tilde{b}(\Delta)$	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.

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

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

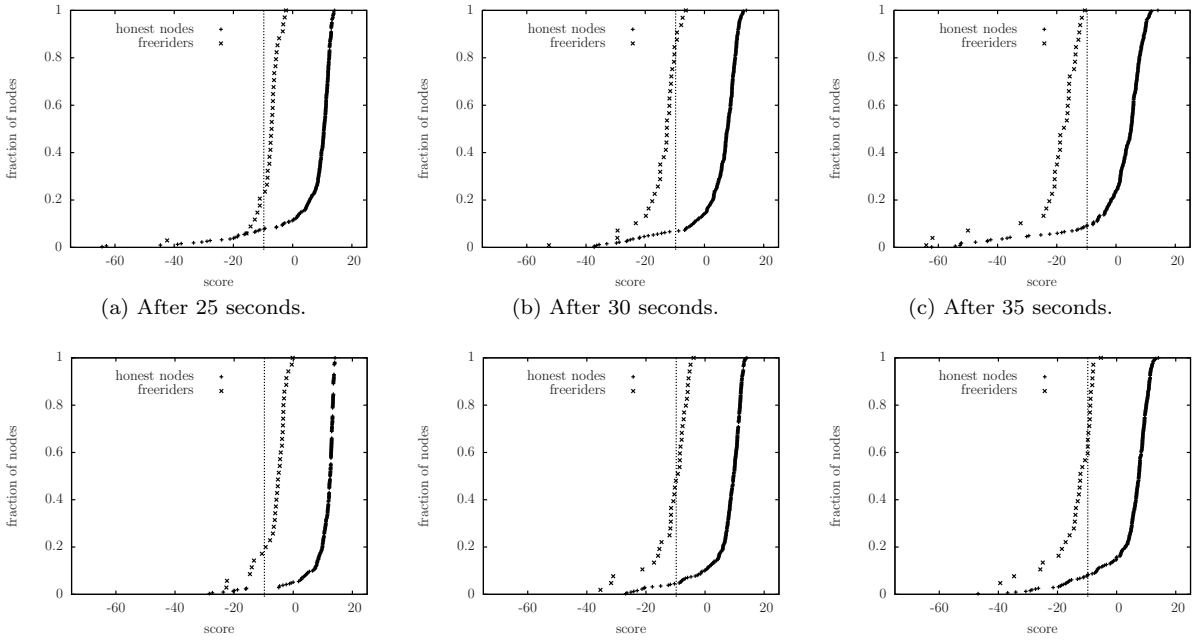
	cross-checking and blaming overhead		
$\mathbf{p_{cc}}$	0	0.5	1
674 kbps stream	1.07%	4.53%	8.01%
1082 kbps stream	0.69%	3.51%	5.04%
2036 kbps stream	0.38%	1.69%	2.76%

Table 5: Practical overhead

6.3 Experimental results

We have executed LiFTinG with $p_{\text{cc}} = 1$ and $p_{\text{cc}} = 0.5$. Figure 15 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 15b ($p_{\text{cc}} = 1$, 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



(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 15: Cumulative distribution functions of scores with $p_{\text{cc}} = 1$ (above) and $p_{\text{cc}} = 0.5$ (below).

freeriders without the need for any cross-check. This explains why the detection after only 35 seconds with $p_{\text{cc}} = 0.5$ (Figure 15f) is comparable with the detection after 30 seconds with $p_{\text{cc}} = 1$ (Figure 15b).

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 [12] that BAR Gossip presents scalability issues, not to mention the overhead of cryptography.

PeerReview [10] 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 case of malicious participants was considered in the context of generic gossip protocols, i.e., consisting of state exchanges and updates [13]. This system relies on cryptography for signing messages between peers and do not consider 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 [12] 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 inherent 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 Achile's heel [22, 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 incentives.

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

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

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

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

[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.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399