Framework for Live-streaming Adaptive to Selfish Behaviour

João Bruno Silva
joao.roque@ist.utl.pt

Instituto Superior Técnico
(Advisor: Professor Luís Rodrigues)

Abstract. Applications like radio or video broadcast in the Internet require the real-time dissemination of multimedia content to many users, a task known as *live-streaming*. The protocols that support need to be highly scalable in order to serve very large numbers of users. To reduce the costs and avoid bottlenecks at the provider, live-streaming can be implemented with the cooperation of the end-users, that may voluntarily offer some of their computing and networking resources to help in the dissemination process. Unfortunately, not all volunteers are altruistic, and some nodes may have incentives to violate the protocol to increase their own benefit (for instance, by avoiding forwarding some packets to save upload bandwidth). Many mechanisms have been designed to cope with rational behaviour in live-streaming applications, such as monitoring and punishment mechanics to discourage selfish behavior. The cost of these mechanisms is, typically, a function of the ratio of altruistic/rational nodes in the system; the larger the fraction of rational nodes, the more expensive the mechanism is. In this work we are interested in looking for mechanisms that can automatically adapt to the ratio of rational vs altruistic nodes, to minimize the operational costs in the good cases.

1 Introduction

The task of disseminating, in real-time, multimedia content to a large number of users has been coined *live-streaming*[1, 2]. Live streaming is particularly challenging when used to provide the coverage to highly popular events, such as major sport events, concerts, breaking news, etc, due to the extremely large number of users that need to be served. Due to this reason, the protocols that support live-streaming need to be highly scalable. One way of achieving scalability and, at the same time, reduce the costs and avoid bottlenecks at the provider, consists in resorting to peer-to-peer solutions, where end-user offer some of their computing and networking resources to help in the dissemination process. Among these solutions, dissemination protocols based on gossip[3] among peers have emerged as a promising strategy, due to their scalability and robustness to failures and churn[4].

Unfortunately, not all volunteers are altruistic, and some nodes may have incentives to violate the protocol to increase their own benefit[5–7]. In particular, end-used may change the software they run in a way that allows them to

receive the content without contributing to the information dissemination, for instance, to save upload bandwidth. Such users are known as *free-riders*[8, 9] or, when considering game theory principles [10, 11] to analyze the behavior of the system, *rational nodes*. In opposition, nodes that follow the protocol faithfully are named *altruistic*. Many mechanisms have been designed to cope with rational behaviour in live-streaming applications [12–14]. Several of these mechanisms require the deployment of some monitoring mechanisms (to assess which nodes are following the protocol) [15] combined with additional mechanisms that punish free-riders (such as avoiding sending information to those nodes) or mechanisms that prevent non-cooperating nodes to assess the information (by cyphering all messages) [13].

The cost of these mechanisms is, typically, a function of the ratio of altruistic/rational nodes in the system; the larger the fraction of rational nodes that one pretends to tolerate, the more expensive the mechanism is. This creates a dilemma: if a very conservative approach is taken, and very robust mechanisms against rational nodes are used, but in reality most nodes are altruistic, the system is inefficient because a lot of resources are waisted to protect against a threat that does not exist in practice. In fact, gossip based dissemination protocols are are extremely robust to faults, and few rational nodes can be masked even without sophisticated countermeasures. On the other hand, if an optimistic approach is taken, but then many rational nodes exist, the system may collapse. Ideally, one would like to use inexpensive mechanisms when few rational nodes exist, and more expensive mechanisms when the fraction of rational nodes is large. In this work we are interested in looking for mechanisms that can automatically adapt to the ratio of rational vs altruistic nodes, to minimize the operational costs in the good cases.

The rest of the report is organized as follows. Section 2 briefly summarizes the goals and expected results of our work. In Section 3 we present all the background related with our work. Section 4 describes the proposed architecture to be implemented and Section 5 describes how we plan to evaluate our results. Finally, Section 6 presents the schedule of future work and Section 7 concludes the report.

2 Goals

This work addresses the problem of designing monitoring and punishment mechanisms, that can deter free riders in gossip-based live streaming protocols, that can adapt to the actual ratio of altruistic vs rational nodes that exist in the system. More precisely, our work aims at:

> Design, implement, and evaluate an adaptive gossip-based live streaming protocol that can adjust its behaviors according to the observed ration between altruistic and rational nodes, in order to minimize the signaling costs and maximize the amount of CPU and bandwidth available to disseminate the streamed content.

In order to achieve the goal above, our algorithm will combine ideas from different systems that have been designed to tolerate free-riders in dissemination protocols. As it will be clear in the related sections, several mechanisms have been implemented in the past, with different costs and tolerating different fractions of rational nodes. Our idea is to incorporate an adaptive mechanism in the algorithm such that nodes can seamless change among two or more monitoring mechanisms, increasing the monitoring costs only when the perceived number of rational nodes is larger, and reducing the monitoring costs when most nodes are altruistic.

The project will produce the following expected results.

> *Expected results:* The work will produce i) a gossip based lightweight protocol; ii) a specification of the protocol; iii) an implementation of the protocol for the Peersim simulator[16], iv) an extensive simulation-based comparative experimental evaluation of our protocol against other protocol from the literature.

3 Related Work

In this section we survey the most relevant works that address peer-to-peer dissemination of information in the presence of both altruistic and rational nodes. We start by discussing, in Section 3.1 a number of mechanisms that have been used to address rational nodes in distributed system. Then, since we are concerned with systems that can support live-streaming to a large number of nodes, it is generally impossible for a single peer to have full knowledge of all the other peers. Instead, these systems often rely on a *membership protocol*, that provides to each peer a partial view of the system. Therefore, for self-containment, we present two membership protocols in Section 3.2. Subsequently, in Section 3.3, we describe different dissemination protocols. As it will be seen, there are two main approaches to implement the dissemination process. The first consists in building some form of spanning tree. Tree-based protocols are more resource efficient but also more fragile: faults or free-riders can easily disrupt the tree. This can be mitigated by using multiple trees in parallel at the cost of increasing the maintenance costs. Another approach consist in using epidemic dissemination[17]. In the later approach, each node forwards each packet it receives to a subset, selected at random, of all nodes in its partial view. As it will be seen gossip-based protocols achieve better robustness at the cost of larger redundancy (and therefore, high bandwidth utilization).

3.1 Addressing Rational Behavior

As we have discussed before, in many systems where end-users have opportunity to alter the behavior of the software, such as peer-to-peer file sharing applications, free-riders tend to appear. It has been studied that, if the fraction of altruistic nodes follows below a given threshold, free-riders tend to dominate

the system and prevent it from providing the intended service[8]. Therefore one needs to augment the system with mechanisms to detect and punish free-riders, for instance, by expelling free-riders from the system. The goal of such mechanisms is to prevent free-riders from dominating the systems, to provide incentives for rational nodes to follow the protocol, and also to ensure that nodes that follow the protocol have no incentives to deviate from their correct behavior. In fact, if a rational node that is following the protocol finds that it is contributing more than most of other noes, it may be willing to change its behavior, and also reduce its own contribution. This may create a cycle, in which the performance keeps on decreasing until the system comes to an halt. In the next paragraphs we introduce some of the main mechanisms that have been proposed to cope with rational nodes.

Payments One possible mechanism to foster cooperation is to implement some payment scheme [18, 19]. Every time a node provides a service to another nodes it gets some payment in return. In turn, a node that has provided services in the past can use it to later request services from other nodes. The currency can be completely virtual or have some correspondence to some real money, corresponding to real payments that end-users make to benefit from the service they get. In this way, nodes that do execute the protocol faithfully, i.e., that provide service, are rewarded and free-riders are quickly depleted of currency. Unfortunately, a payment-based system is not trivial to implement. First, it is necessary to ensure that nodes cannot fabricate currency, and this usually involves the use of complex cryptography techniques. These systems also require some accounting module to securely store each peer's virtual currency and a settlement module that enforces a fair service in virtual currency exchange. All these components are complex to implement, and often require the use of a a secure central entity that all peers would trust.

Reciprocity [20] In a reciprocity-based system, a peer monitors other peers' behavior and evaluates their contribution based on their past actions. There are two types of reciprocity-based schemes: direct and indirect.

Direct-reciprocity schemes In direct-reciprocity schemes, Alice decides if she will share her information with Bob, based on the service that Bob has provided to her. BitTorrent [21] is a famous example that uses a direct-reciprocity scheme, that uses a tit-for-tat incentive mechanism in order to a user find the best set of peers that are willing to contribute.

Indirect-reciprocity schemes In indirect-reciprocity schemes, Alice decides if she will share her information with Bob, based not only on the services that Bob has provided to her, but also on the service that Bob has provided to others users in the system. Intuitively, indirect-reciprocity schemes are harder to implement and are more expensive as they require more messages' exchange in order to make a decision. Another drawback in this approach is the fact that it is vulnerable to

collusion behavior: a set of peers may be colluding and will always give positive feedback to others about users within the set.

Reputation [22] can be seen as a form of indirect reciprocity, where a numerical value, the node's reputation, captures the past behavior of a node as it is seen by its peers. Reputation is usually only meaningful after a reasonable amount of interactions among nodes, such that temporary faults outside the control of the node (such as a temporary network outage) do not affect negatively the reputation of altruistic nodes. The main difference from reputation and indirect reciprocity is that information about peers reputation is maintained and the system, itself, will provide better download capacity to nodes with higher reputation [23]. Reputation schemes can be local or global.

Local Reputation In local reputation, peers store, locally, information about others peers that they've interacted with. This is the simplest approach as it does not required information to be shared and each peer can have different reputations levels to different peers.

Global Reputation In a global reputation approach, the reputation of each peer is based on the information obtained from a set of peers. Although this approach may seem more accurate, it creates a set of problems. First, it is necessary to take into account that a rational peer may lie about the reputation of other nodes. Second, sharing information requires additional message overhead and it may be necessary to use a central authority to store and manage reputation information, which is difficult to implement in pure peer-to-peer networks.

Reputation mechanism also assume that peers' identities should be preserved across sessions. This creates the challenge to not only uniquely identify each member of the system, but also store this information in a reliable node. As being said, this is difficult to implement in peer-to-peer systems and can compromise scalability.

Joining the System One problem with reciprocity and reputation systems is that newcomers have no "history". On one hand, one would like that the procedure for joining the system is not extremely expensive and or slow, such that new contributors are not discourage to join the system. On the other hand, if joining the system with a new identity is very simple, this creates an opportunity for free-riders to escape the monitoring mechanisms by repeatedly switching to a new identity (i.e., whitewashing). A trade off is to impose some simple but efficient penalty to newcomers, such as a delay on startup. This time most be small enough to incentive newcomers to join the system and long enough to not be worth to exploit it by selfish users.

3.2 Membership Protocols

In Gossip based protocols, the reliability depends on sending messages to random members of the system, however, gossip's systems can have countless

members which makes very difficult to provide a list of all members to every peer in the system, specially because nodes are free to join the system or to leave it. This would compromise the scalability of the system as nodes would spend more time trying to get the most updated list of members instead focusing on disseminating information. In practice, members of a gossip system only store a partial view of the system and may, periodically, update it by exchanging it with another member. The partial view should provide a good sampling of the system in order to a member be able to select, at random, a subset of member that will receive information. One important aspect of gossip protocols, is how large this random subset should be in order to guarantee that every member in the system receives the information. In [24] was proved that if there are n nodes in the system and if every node gossips information to, at least, log n, the probability of every node receiving the information is close to one. In this section, we will describe two memberships protocols for gossip systems.

SCAMP [25] is a reactive membership protocol for gossip-based peer-to-peer systems. The purpose of the protocol is to maintain, at every peer, a partial view of the system, which approximates a random sample of the entire system membership. One of the most interesting aspects os SCAMP is that the size of the partial view is not fixed; instead, it is proportional to the size of the system, even if each individual node has no explicit knowledge about the total number of members that exist in the system. However, from the operation of the join procedure nodes can indirectly infer when they need to enlarge their views to accommodate more nodes in the system.

The protocol works as follows. A new node must first obtain the contact of another node already in the SCAMP network. This is achieved using a mechanism orthogonal to the protocol (for instance, using localized flooding or pre-defined trackers). Then, the new node starts a join procedure by sending a join request to the contact node. When the contact node receives a new subscription request, it forwards the subscription request to all members of its own local view. It also creates additional copies of the new subscription that are forwarded them to randomly chosen nodes in its local view. When a subscription request is received by another node it can either be accepted (with probability p) or forwarded to some other random node (with probability $1-p$). Every time the subscription request is forwarded a time-to-live filed is decremented. When this filed reaches zero the request is deterministically accepted and the size of the local view is increased. In order to leave the network, a node must flood an unsubscribe message to the entire system. If a node becomes isolated, this means that its identifier is not present at any local view, it will resubscribe through a node in its partial view. A node may become isolated if all nodes that contained it on their partial view leave the system, either by crashing or unsubscribing.

HyParView [26] One disadvantage of the SCAMP protocol is that it does not implement any form of failure-detection mechanism other than the periodic garbage-collection/re-subscription described above. This requires the use of large

views to keep the network connected and also makes the partial views highly unstable, as they are periodically refreshed. This makes hard to use the membership service for building protocols that require stable connections among peers, such as tree-based dissemination protocols. HyParView[26] is a membership protocol designed to eliminate these disadvantages.

HyParView operates by maintaining two complementary views, namely a *passive view*, that provides a good sample of the system (the passive view can be maintained by a protocols such as SCAMP), and an *active view*, a smaller subset of views that are continuously monitored and quickly replaced if, but only if, they have failed. Furthermore, unlike SCAMP partial views, active views in HyParView are symmetric. This ensures that nodes maintain a fixed out-degree but also a fixed in-degree, and cannot be easily disconnected from the network.

The active view are maintained has follows. As in SCAMP, a new node must send s subscription request to a node already in the network. The node that received the join request will add the new node to its active view. Then, it proceeds to forward the request to every other member of its active view. As in SCAMP, subscription request perform random walks in the network until they are eventually accepted. However, unlike in SCAMP, when the join is accepted, a bi-directional link is created with the joining node (to ensure view symmetry). This kink is used by each peer to actively monitor the other nodes in its active view. If one of these nodes is detected to be failed, a candidate for replacement is picked form the passive view. In fact, the role of the passive view is only to serve as a pool of good candidates to replace failed nodes in the active view. As a result, is is possible to maintain the network connectivity with active views that are short and stable.

3.3 Dissemination Protocols

Altruistic Dissemination Altruistic dissemination considers the model when every node in the system is altruistic, this means that it follows the specified protocol and it is willing to disseminate information. We can find three different types of organizations to achieve this: Basic Gossip, Trees and a BitTorrent fashion way.

Basic Gossip, also called epidemic protocols, is very appealing in large scale distributed applications. These protocols are popular because their possibility to be fully decentralized, their ability to reliability pass information among a large set of nodes, even if some of them crash or if messages are lost. This characteristics fit, perfectly, the peer-to-peer paradigm. In a gossip-based protocol, each member of the system exchanges information with a subset of its neighbors. Nodes should select for who they disseminate information using a random sample of all nodes in the system. Basic Gossip protocols have very little overhead necessary to maintain its structure. Nodes just need to store a sample of members that are in the system, eventually, update this list (if churn is taken into account). Since information is spread in a random way, nothing prevents a node to receive the same information multiple times from different nodes, however, it is exactly

this redundancy that provides gossip protocols the robustness to support lost messages and nodes leaving the system.

Trees may be the most intuitive and natural way when thinking about disseminate information that, in our case, always have the same origin, the streamer. The most simple approach creates a tree where the streamer would be the root and it would disseminate information for a subset of peers that forward it to their children. If we think about simple trees, where each peer only has one parent, instinctively, this approach has almost no redundant message. This happens, because, contrarily to Basic Gossip, nodes would receive information only from one node, their parent. Considering an heterogeneous network, where peers can have different upload capacities, trees offer the possibility to put nodes that have better upload capacity at the top, while nodes with low upload capacity as leafs. This organization would contribute not only for a better availability but also for a better resource management. Trees, however, suffer from two main drawbacks. First, finding the best tree organization may not be a trivial process and would require time and computational cost. Second, it is hard to handle churn. If a node that is positioned on a top position leaves the system, a major part of the nodes can stay, until the tree is reconstructed, without receiving any content. When a new node joins the system, the simpler approach, would select any leaf to be its parent. However, this can lead to poor resource management.

BitTorrent [21] is a file-download protocol. Each peer is responsible for maximizing its own download rate by finding the best set of peers that are available to share information. If a peer has a high upload rate, it will be able to maximize its download capacity. Contrary to Gossip protocols, where nodes upload and receive information to and from random members, and to tree, where the system organizes its structure, in BitTorrent, a node has the responsibility to find the set of neighbors that will provide him the least download time. BitTorrent cuts files into pieces and every piece is divided into multiple sub-pieces. In order to minimize download times, BitTorrent makes use of a set of rules. *Strict priority* enforces that once a sub-piece was requested, a node must request the remaining sub-pieces before sub-pieces from another piece. *Rarest first* allows peers to start downloading pieces which the fewest peers have first. *Random first piece* offers the possibility to peers who just started the download and have nothing to upload to get a complete piece. Pieces to download are selected at random until the first complete piece is received, after the strategy changes to rarest first. *Endgame mode* allows a peer, when all sub-pieces that it does not have are already being requested, to request all sub-pieces to all peers. Later, it cancels the sub-pieces that arrive in order to do not waste too much bandwidth.

Dissemination with rational nodes In reality, not all nodes are altruistic [5]. While it is desirable that every node follows the protocol, some nodes either by poor upload capacities or intentional bad intentions will not contribute for content dissemination. In simple tree topologies, a single node can crash a major

part of the system if it starts to block outgoing communication, specially if it is a node that is positioned of the first levels of the tree. Basic Gossip protocols are also vulnerable to selfish behavior, however due to its robustness and redundant messages system, it tolerates a higher percentage of selfish users than trees. In this section we will describe systems that aim to prevent selfish behavior in Gossip based protocols [13–15], Tree and what strategies BitTorrent has to achieve the same goal.

BAR Gossip [13] was the first peer-to-peer data streaming application that aimed at offering guaranteed predictable throughput and low latency in the BAR (Byzantine/ Altruistic/Rational) model. The main characteristic of gossip protocols is that each node exchanges data with randomly selected peers. This randomness gives gossip protocols their enviable robustness. However, rational users can use this randomness to game the system and hide selfish actions. This happens because if there is no determinism, it is very difficult to argue that a node is sending information to the incorrect set of nodes. In order to solve this problem, BAR Gossip relies on a verifiable pseudo-random partner selection to eliminate non-determinism. In BAR Gossip, the streamer relies on two protocols to disseminate information: Balanced Exchange and Optimistic Push.

Balanced Exchange The Balanced Exchange mechanisms allows clients to trade updates one-for-one, where each party determines the largest number of new updates it can exchange while keeping the trade equal. A big drawback in Balanced Exchange is the overhead added by encrypted content that is sent in order to make sure that both users act faithfully. Another problem is that if a node has no valuable information to give, the trade will not happen.

Optimistic Push The Optimistic Push mechanism provides a safety net for clients who have fallen behind by allowing clients to obtain missing updates without giving back a set of updates of equivalent value. In order to make optimistic push fair and to avoid that users use it to hide selfish behavior, a user who has fallen behind has to send junk data when trading updates. To avoid abuses, the size of the junk data that a member has to send must be bigger than the size of actual information that is received. This tries to prevent Free Riders from using it as way to get information only by sending junk. This strategy has an obvious drawback: junk data is not useful and it his consuming bandwidth of both sender and receiver.

BAR Gossip is also able to tolerate byzantine behavior by requiring every node to sign their message using their private keys to provide authentication, integrity, and non-repudiation of message contents. Overall, Bar Gossip is able to provide a good service even when all clients are selfish or when there are less than 20% of Byzantine nodes. These properties are achievable by using heavy communication protocols, more concretely by ciphering every message. This is clearly an unnecessary overhead if the system is not composed by enough rational nodes that can warm the quality of service.

FlightPath [14] can be seen as an evolution of BAR Gossip. One of the problems of using random gossip to stream live data is the widely variable number of trading partners a peer may have in any given round. High numbers of concurrent trades are not desirable for two reasons. First, a peer can be overwhelmed and be unable to finish all of its concurrent trades within a round. Seconds, a peer is likely to waste bandwidth by trading for several duplicates updates. To address this problem, FlightPath distributes the number of concurrent trades more evenly by providing a limited amount of flexibility in partner selection. This is achievable by making reservations: A peer c reserves a trade with a partner d before the round r in which that trade should happen. Reservations are effective in ensuring that peers are never involved in more than 4 concurrent trades.

In basic gossip trading protocols it is desirable to disseminate the most recent updates over older ones to spread new data quickly. However, in a streaming environment, peers may sometimes value older updates over younger ones, for example when a set of older updates is about to expire and the peer do not have it yet. The drawback in preferring to update old information is that the received information may not be useful in future exchanges because many peers may already have it. Flightpath provides a peer with the possibility to receive updates from older rounds first and then updates in most-recent-first order. One weak point of BAR Gossip was how Optimistic Push was designed. Flightpath solves this problem using the Imbalance Ratio: Each peer tracks the number of updates sent to and received from its neighbors, ensuring that its credits and debits for each parter are within a certain threshold of each other. This is a similar strategy to the one we will use to compute the nodes score. The final improvement that Flightpath offers compared to BAR Gossip is the Trouble Detector. Each peer monitors its own performance by tracking how many updates it still needs for each round. If its performance is low, then that peer can initiate more trades in order to avoid missing an update.

LiFTinG [15] detects Free Riders in Gossip protocols with asymmetric data exchanges. Asymmetric data exchanges happens when a node is able to send more data to another node than it receives from him. LiFTinG, in order to track Free Riders, requires nodes to track others nodes behavior by cross-checking the history of their previous interactions and relies on the fact that nodes disseminate information to a subset of random nodes. This property prevents colluding behavior, where nodes can chose to send information always to the same subset of nodes or nodes covering each others' bad behavior. A node checks another node behavior according to a probability that is a constant in the system.

LiFTinG works in a generic three-phase Gossip protocol where data is disseminated following an asymmetric push scheme. Three-phase Gossip consists in informing a node of the available information, then the receiver sends back the list of identifiers it wants and finally the first node send him all requested information. Symmetric push requires additional steps in order to negotiate the amount of packages that both nodes will exchange.

LiFTinG makes use of a deterministic and statistical distributed verifications procedures based on the node's past interactions. Deterministic procedures try to validate that the content received by a node is later propagated by the same node. Statistical procedures check that the interactions of a node are evenly distributed in the system using statistical techniques, this means that a node should disseminate information to a random set of peers and should not collude by choosing to who it sends informations.

LiFTinG does not rely on heavyweight cryptography and incurs only a very low overhead in terms of bandwidth. It is also fully decentralized as nodes are in charge of verifying each others' actions and monitoring each others' behavior, without the need of a dedicated server.

Detection of Free Riders is achieved by attributing a score to every node. If a node's score decreases below a threshold, it is assumed that it is Free Riding and the system punish it. When a node detects that some other node is Free Riding or is not faithfully following the protocol, it sends a blame message containing a value against the suspected node. Summing up the blames values of a node results in a score. Each node is monitored by a set of nodes, called managers. Managers are distributed among participants in order to avoid undesirable cooperative behavior and they collect blame messages against the nodes they monitor. When the score of a node drops beyond a threshold, the managers spread - through gossip - a message to inform users. Users then remove this node from their membership, consequently that node stops being part of the system as it will not receive any information. In the following subsections, we will describe the procedures that LiFTinG uses in order to detect Free Riders.

Direct verifications There are two direct verifications. The first aims at ensuring that every chunk of information that a node requests is served. Nodes cannot arbitrary request information, however if some node, in the first step of the gossip, purposes a certain set of chunks, then the receiver has the right to ask for every chunk it requires. If any node refuses to send a chunk asked, the requesting node blames the proposing node with a score that is proportional to the amount of chunks not delivered. This detection can be done locally and it is therefore always performed.

The second verification checks that receive chunks are further proposed to another nodes within the next gossip period. This is achieved by a cross-checking procedure that works as follows: Alice who received a chunk C from Bob acknowledges to Bob that she proposed C to a set of nodes. Then, Bob sends a confirm request to the same set of nodes to check whether they effectively received a propose message from Alice containing C. If Alice refuses to send Bob the acknowledge message, she will be blamed according to the amount of nodes that she should disseminate the information to. Alice will also be blamed for each missing or negative answer message that Bob receives from the set of nodes that he contacted.

Verifications a posteriori The random choices made in the partners selection must be checked, this is necessary to avoid collusion. To verify it, a node re-

quests the local history of a node. This local history has the last interactions that a node did during the last seconds. Since the partner selection is random, it is possible to use statistically calculations to understand if a node is try to use a non-random algorithm to compute it. In order to confirm that a node send the correct history and did not change it, the verifier will then pick a subset of nodes that are present in this history and request a confirmation that validates its integrity. If the node sent a false history, his score will be affected according to the amount of nodes who do not validate its history. LiFTinG incurs a maximum network overhead of 8%. When Free Rides decrease their contribution by 30%, LiFTinG detects 86% of the Free Riders and wrongly expels 12% of honest nodes. False positives were nodes that their actual contribution was smaller than required. However, it was because poor capabilities, as opposed to Free Riders that deliberately decrease their contribution.

LiFTinG assumes that a selfish node will not send false blame messages against an altruistic node. There are two reasons for this. First, altruistic nodes are the main reason for information to be disseminated. Removing these members from the system may decrease the information that its available for all nodes, including selfish users. Second, blaming a selfish node does not guarantee that a selfish user will not get caught by another users.

FOX [12] is a tree based system for live streaming that assumes that all peers are greedy. In addiction, it provides theoretically optimal download times when everyone cooperates. In a simple way, FOX protocol builds multiple sub-trees being the streamer the root of all of them. The streamer disseminates to every tree a different subset of the information. Later, each parent node of the subtrees proceeds to spread the information to its child until it reaches the leaves. The leaves then exchange their information with the leaves of others subtrees in order to have the whole set of updates. After this, leaves send the new information back to their parents. And thus, every node is the global tree receives every update. If selfish nodes complete their downloads at different times, the system runs the risk of last-block collapse, in which nodes begin dropping out of the system when they finish, leaving the remaining nodes with no help to finish their downloads. To prevent this, FOX enforces participation by augmenting the block exchange with a novel encryption algorithm. This algorithm requires nodes to exchanges decryption keys by sending one bit of the key at a time without letting the other get ahead by more than one bit. In the worst case, one of the nodes in the exchange cloud get the final bit and leave the system, but since their are exchanging bits, the remaining node need only to try both values. In essence, all participants finish their downloads simultaneously. A major drawback in FOX is the assumption that the system is not dynamic. After the system structure is defined, every node from root of the subtree to the leave cannot leave or crash. This is a strong assumption that in practice can compromise the whole system if it is not verified.

Coolstreaming [27] uses a hybrid pull and push mechanism, in which the video content are pushed by a parent node to a child node except for the first block. This helps to significantly reduce the overhead associated each video block transmission, in particular the delay in retrieving the content. Coolstreaming makes use of multiple sub-streams scheme is implemented, which enables multi-source and multi-path delivery of the video stream. Observed from the results, this not only enhances the video playback quality but also significantly improves the effectiveness against system dynamics. Coolstreaming has two main disadvantaged. First, it does not considered the Free Riding problem. More than that they estimate that around 65% of the users do not contribute for the content dissemination. Second, Coolstreaming places strategically a number of servers that help in content dissemination and to reduce the start up time. Although the use of dedicated servers can help a lot the system, either by helping uploading content or rely on them to analyze the state of the system, they contradict the whole purpose of P2P systems.

BitTorrent [21], a file-sharing system, files are split up into chunks and the nodes who want to get a file cooperate in a tit-for-tat-like manner. This strategy aims to try to prevent parasitic behavior as a node is more likely to cooperate with a node that had cooperate with him in the past. Each peer will try to maximize its own download rate by contacting the best set of peers who are willing to cooperate. Contrarily to live-streaming, in file-sharing, when a peer finishes downloading any file, it may become an extra dissemination source. This behavior is the opposite of Free Riding; a user will only disseminate information without expecting anything in return. In live-streaming this is not achievable. Because the content of the stream is dynamic, so a node never has the full information so to keep disseminating, it has to keep receiving.

BitTorrent has two types of up-loaders: seeders and leechers [7].

Seeders are nodes that already have successfully download a file and provide its content to another nodes. Seeders upload information to all peers in a round robin fashion way.

Leechers are nodes that are still downloading the file. Leechers upload information mainly to peers are able to provide some pieces of the file in return.

BitTorrent also uses a mechanism called "Optimistic Unchoking". Using this mechanism, a peer will reserve part of its upload bandwidth to send pieces to random peers hoping that it will discover better partners and to increase the possibility for newcomers to be part of the system. This mechanism is vulnerable to white-washing (a node leaving and re-entering the system with a new identity) and a node can abuse this strategy to download the file and minimize as much as possible his contribution. This is a major difference between the peer-to-peer file-sharing and live-streaming paradigm. In live-streaming paradigm, the node was little or no incentive to white-wash because even with a low startup time,

the node will lose information. This property give us some flexibility to create a protocol that requires some time to startup.

According to BitTorrent terminology, in live-streaming application there are no seeders. This is because new information is being constantly generated and peers may not persistence store the information they receive. From one side, this property makes information more 'valuable' as nodes have a limited time to access it. However, this short time is why it is so important that every node follows the protocol.

3.4 Discussion

A compairison of the systems described above is preseted in Table 1. Under perfect conditions, where there are no Free Riders and nodes do not leave nor enter the system, tree based approaches offer a natural and efficient structure to spread information. Also, in the best case, no nodes received repeated information which provides a minimum number of messages on the network. However, as we mention, perfect conditions are impossible to achieve in practice and therefore, trees suffer from nodes dynamics and thus are subject to frequent reconstruction, which incurs extra cost and renders to sub-optimal structure. Another big drawback of the tree approach is that the higher the node is in the topology, the higher its responsibilities will be and as we mention earlier, it is impossible to rely on any node besides the streamer, so if this node decides to have a malicious behavior, a major part of the system can be compromised. This results in a poor resource usage and unfair contributions in that a leaf node cannot contribute by disseminating information. The multi-tree approach has been proposed to cope with problem. Multi-trees achieved two improvements. First, better resilience against churn. Second, fairer bandwidth utilization from all peer nodes. However a multi-tree scheme is more complex to manage in that it demands the use of special multi-rate or/and multilayer encoding algorithms. For these reasons, we will adopt a Gossip like protocol.

	Tree	Gossip	FR Prevention	FR Detection	Overhead
Basic Gossip		✓			Very Low
BAR Gossip		✓	✓		High
FlightPath		✓	✓		High
LiFTinG		✓		✓	Medium
FOX	✓		✓		Medium
Coolstreaming		✓			Low
BitTorrent		✓			Very Low

Table 1. Compairison of systems

The reason why overhead is so high in BAR Gossip and FlightPath is because these systems prevent byzantine behavior. They assume nodes can modified information and spread it. To detect an prevent it, messages are encrypted with the streamer public key. Systems with low or very low are vulnerable to Free Riding. FOX is able to tolerate Free Riders, however a single byzantine node can collapse the entire system and the system's incentive structure depends on the fear of mutual destruction. LiFTinG identifies Free Riders and expels them from the system.

4 Architecture

Peer to peer for live stream is an architecture not complete decentralized because there is a node (the streamer) that will be the source of all the packets, this node can be a seen a traditional server. This provides two important properties: First, because this node is the initial point of the information dissemination, it is safe to assume it never crashes. (If it crashes, the whole stream will be compromised, so as long it does not fail due a system related cause it is possible to rely on his availability). Second, as the main information disseminator and owner of the stream, we can assume that this node will strictly follow the protocol under any circumstances.

Secondary nodes, all nodes expect the stream, expect a minimum quality of service T. Quality of service is based on the amount of time that a user can watch the stream without any content flow break. Quality of service depends on three major attributes: Node integration, node download bandwidth and state of the system.

- Node integration is the relation between the probability of all nodes sending information to that node versus the actual information that it receives.
- If a node was a low download bandwidth, its quality of service can be low simply because it cannot get the contents faster than its reproduction time.
- State of the system considers the amount of user that are Free Riding, thus warming the system.

Nodes integration is solved by gossip protocols that prove that if every node disseminates information to, at least, log(N) users, being N the total number of users, then every node has a probability near 100% of receiving the information. We assume that a node must have enough download bandwidth to be able to receive all information needed to reproduce the stream. How to maintain a good system state and reduce the impact of Free Riders will be the focus of this work.

4.1 Common specifications

In this sub section, we will specify the elements that are the same for both Optimistic and Pessimist behavior.

Membership Protocol As being said, one property of peer to peer system is its highly scalability. In order to provide this characteristic, it is necessary to handle node dynamics. This means that nodes can join the system and leave it, either by crashing or free will. Due to its simplicity, we will adopt a protocol similar to SCAMP to be our membership protocol. Even tough HyParView was able to have better results than SCAMP in terms of reducing redundant messages, our system can benefit from it. The motivational for this is that the more message on the network the higher the probability that every node has to receive the information. However, it is good practice not to abuse this as it requires additional work for each altruistic peer and the system can collapse because nodes may not have enough computational resources to provide information for a huge amount of peers.

Join Mechanism When a new user joins the system, he does not know any other user and more important than that, he has no useful information to disseminate. The first thing it will do is notify the streamer and ask for a partial view of users. The next time that the streamer disseminates information, he will include the identification of the new user on it. After this moment, the new user can start to be part of the system and when it receives the first chuck of information it can start to disseminate it through the members of its partial view.

Partial View As previous said, every node will have a partial view that contains identifiers of the members of the system. The size of this list must be big enough for a node to have variety when calculating for who it should disseminate information, but small enough to make the system able to scale. Partial list will be split into three lists: Active list, Back up list and Black list.

Active list has the nodes identifiers for whom a node should mainly disseminate information. A node will not upload information to every member of this list, but instead only for a sub set of it.

Backup list contains members that will be promoted to the active list when its size decrements, which can happen due nodes leaving the system or being moved to the black list. Eventually, a node will disseminate information to some members of this list, hoping it will find more altruistic nodes.

Black list contains the nodes that are believed to be Free Riders.

A node can disseminate information to any member of its Partial View, however, it will prioritize the active list and will send information to a member of the black list with very low probability.

Tit-for-Tat Similarly to BitTorrent, we will use a tit-for-tat mechanism. Every node will store a list that is a pair <Score, Id>, where Score is a number that represents the score of the Id node. This score is updated by -1 or 1 every time a node sends or receives information to or from another node. In order to make the system scalable [28], each peer will store a partial list that will contain a sub set of all peers in the system. Nodes should disseminate information to a sub set of nodes that are members of this partial view and, respectively, update their scores.

The Tit-for-Tat mechanism that we aim to implement has two goals. First it will provide nodes, who contribute, to increase their changes of receiving information. Second, it will be a simple method to avoid sending information to Free Riders. Since nodes will send information with higher probability to the ones who had sent them information in the past, less information will be available for the Free Riders. This situation is desirable because altruistic nodes will keep receiving all updates and Free Riders will, eventually, starve. The information necessary to implement the TfT strategy will be stored on the Score List.

Score List The score list will contain information about the previous interactions that a peer had. This means that when a peer uploads or receives information, it must update the list. Since a node updates its score list when information is send or received, nodes are able to calculate for whom they should send information and from whom they have higher probability to receive information. The main goal of this list is not only for a node to be able to identify the best set of peers that are willing to exchange information but also to identify the nodes that are not contributing for the dissemination of information, which are the ones that a node expects to receive information from, but never does. Intuitively, if a node follows the protocol, it will have more users who are available to send him information and therefore its integration increases. However, a Free Rider, eventually, will collect negative scores among all users and his integration will be reduced.

Due to the randomness of gossip protocols, it is possible that a node sends information to another peer multiple times without receiving anything in return. In our protocol, this would result in the receiver node being considered a Free Rider, even if it was not. To avoid this problem, nodes will try to upload information, with a probability near 100%, every time he receives new information to nodes who score is close to limit. This is achievable due the symmetry between scores. A node cannot force another to send him information, so in the extreme case, a node can only fake his only local scores and pretend that every user should send him information. In fact, this is the Free Rider strategy. Where instead of blocking every outgoing connection, a node can fake his score by pretending that he is the only altruistic node in the system and therefore he does not have to upload any information. For this reason, a node is not able to exploit the score list and increase the probability it has to receive information from others.

4.2 The Optimistic Behavior

The optimistic behavior assumes that a major part of the nodes is able to receive all information necessary to reproduce the stream without breaking the content flow, this means they received all sequential packages. The optimistic behavior will be built on top of traditional gossip.

One important goal of the Optimistic behavior is to minimize the overhead necessary to disseminate information. However, it will be more expensive than basic gossip as nodes will maintain the score list, will have to calculate the best set of peer to disseminate information, instead just random selecting peers. In terms of message overhead, nodes will have to perform cross checking validations which will increase the amount of messages on the network that do not contain streaming information. Eventually, nodes will also to disseminate their blacklists, which contributes for additional message overhead.

4.3 The Pessimist Behavior

The pessimist behavior must ensure a good quality of service to every node independently the percentage of Free Riders. Intuitively, this protocol has to provides one the two following properties. First, it can force Free Riders to contribute for the content dissemination, meaning that they leave the system, re-enter and start to follow the protocol. This would allow the system to be constituted only by altruistic users. Altruistic nodes, who act by following the specified protocol, would disseminate the information among each other and information would reach every user. Second, it can prevent altruistic nodes from sending information to Free Riders. The problem of Free Riders is that when information reach any of these nodes it stops being disseminated. If we avoid information to be upload to this nodes and instead we send it to members who will also contribute, in practice, the system will have no Free Riders. This prevention can be done by spreading a blacklist that contains the identifiers of Free Riders.

The Pessimist behavior that we would use will be based on LiFTinG [15]. One improvement that it is possible to make is adjusting the probability when a node confirms the behavior of another based on the score that the node has. This means that if Alice has a positive score attributed to Bob, she can assume that Bob is disseminating information not only for but also for others, so she will confirm his behavior less often. However, if Alice has a negative score attributed to Bob, she may assume that Bob is refusing to spread information, and in this situation, she will check Bob's behavior with higher probability.

One particular characteristic about LiFTinG is that its mechanisms do not aim to force peers to contribute, but instead it identifies who is not contributing. This is a very interesting property that will allow us to apply punishments directly to a set of users. In addiction, if our optimist protocol is able to identify with success Free Riders, then we can immediately punish them. Note that our simple protocol does not need to identify all the Free Riders: detect only a

subset of it is enough to apply a immediate punishment as long as there are no false-positives.

The Pessimist behavior will also use mechanisms present in FlightPath. Nodes will disseminate information symmetrically, this means that instead sending all information requested, they will negotiate the maximum about information that both parties are available to exchange. The reason for this is to maximize trades with altruistic nodes and minimize information that is sent to Free Riders. However, if an altruistic node has fallen behind or it just joined the system, it may have no useful information to trade. Similarly to FlightPath, we will use the *imbalance ratio* where nodes can perform asymmetric exchanges as long as they update their credits and debits, in our case, the score list. However, when performing an asymmetric exchange, we will always perform LiFTinG verifications: direct to ensure that a node send all information it proposed. And cross checking to ensure that a node proposed to other the information it just received.

The Pessimist behavior will combine mechanism both from LiFTinG and FlightPath. However, since prevention of byzantine behavior is out of the scope of this project, we will not use the cryptographic mechanisms that FlightPath enforces to ensure that message content is not modified.

Although it is always preferred to run a simple and lightweight protocol, since altruistic nodes to not try to deviate the system's rules, they will run any protocol that is required. Free Riders, from the other side, will try to find a way to avoid contribute. One important aspect to considered is the fact Free Riders will be the main responsible for protocol changes, and if it happens they will be forced not only to contribute by disseminating the information but also they will have the extra overhead that the second protocol imposes, which contradicts their original strategy. In real life examples it would be interesting to know what would be their strategy: Or they do nothing keep being punished and, at extreme case, expelled or they leave and re-enter the system with a new identity at the cost of losing information.

4.4 Switching

Users expect a minimum quality of service and if this requirement fails, they will notify the streamer. Streamer will collect notifications from the users and will try to compute the state of the system: If only a small percentage of users send him notifications, it is safe to assume that it is a local problem. This means that nodes may have low download bandwidth or some packets were lost. However, if a high percentage of users send notification, the streamer can assume that there are more nodes Free Riding than the protocol supports. In this situation, the streamer will notify members through gossip and to run the pessimist behavior in order to minimize the negative impact of selfish nodes.

There must be a point where the system can return use the optimistic protocol again. To find this moment we could use a similar process used to identify selfish behavior: if, after a certain time, users keep receiving a good quality of service, it can be safe to assume that the system has possibility to use the optimistic behavior again. Another solution, would consist on the streamer to expel

nodes from the system, by gossiping their identifier and nodes would add it to their black list. Then the streamer can, approximately, estimate the percentage of Free Riders. If it less than the supported by the optimistic protocol, the protocol changes again.

4.5 Discussion

When a Free Rider is spotted by a node, it will be black listed. However, it is possible that the Free Rider has not yet been spotted by other nodes and thus can still receive all information. One intuitive approach would disseminate nodes' black list among peers that are believed to be altruistic and they could merge the received list with the actual. We could also create an indirect reciprocity method where the score list would be disseminated and nodes calculate scores based on the lists they receive.

Finding the moment when the behavior of the protocol should change is not trivial. In our solution, we assume that the streamer has a global vision of the system and based on it, is able to estimate if quality of service drops due selfish behavior. This may creates scalability issues where it is not possible to maintain an updates list of every member in the system. We are aware of this problem and we aim to create a solution that does not compromise the scalability.

Figure 1 represents a diagram of the architecture described.

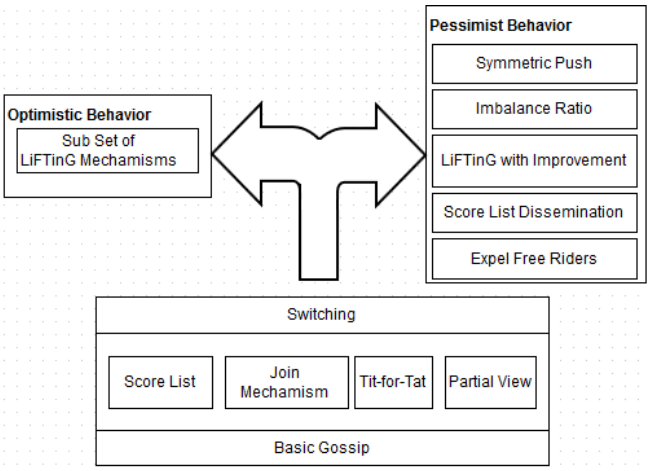

Fig. 1. Diagram of the architecture

5 Evaluation

PeerSim is a Peer to Peer simulator that allows developers to create systems that can consist in millions of nodes, which typically join and leave continuously.

This model fits our protocols and therefore we will use it for evaluation purposes and to be able to trace practical metrics, such as thresholds, lists sizes and times. PeerSim also allows to define how the population is composed, this means that we are able to define the percentage of altruistic and Free Riders nodes that are in the system at a given time.

The Optimistic behavior is expected to tolerate a small percentage of Free Riders. We will make experiments with different percentages of Free Riders and calculate the percentage that is needed to altruistic nodes start to miss information. When a node is calculating to who it should disseminate information it must be done as fast as possible. We will trace the time it takes to perform this computation with different amount of members in its partial view of the system. This calculation in terms on time should not add an overhead higher than 75% compared to basic Gossip. The black list of each node should not contain any false positive. We will be successful if, after running the protocol for a given time, we have no false positives and if every Free Rider is member of, at least, one black list. To achieve the last part, it is necessary that Free Riders do not whitewash.

When running the Pessimist behavior the additional overhead should not be higher than 15% of a protocol that is able to provide a good quality of service in the presence of Free Riders. This includes i) Memory overhead, necessary to keep all the active views; ii) Computational overhead, necessary to calculate the set of members that a peer will disseminate his information and extra confirmations; iii) Message overhead, necessary to run protocol specifications, including verifications and calculations of the best number of updates exchanges.

When using back the optimistic behavior, we will be successful if, with no churn, all members are able to receive a good quality of service. To evaluate it, we will stop node dynamics, and record every node notification relatively to the quality being non-optimal. To be successful, there should be no messages.

6 Scheduling of Future Work

Future work is scheduled as follows:

- January 9 - March 29: Detailed design and implementation of the proposed architecture, including preliminary tests.
- March 30 - May 3: Perform the complete experimental evaluation of the results.
- May 4 - May 23, 2015: Write a paper describing the project.
- May 24 - June 15: Finish the writing of the dissertation.
- June 15, 2015: Deliver the MSc dissertation.

7 Conclusions

Peer to peer live streaming services can be compromised by the presence of selfish users, who refuse to contribute for the content dissemination. Some

techniques, at the cost of message and cryptographic overhead, try to eliminate selfish behavior. However, none of them take into account the state of the system, which means that all overhead is unnecessary when a major part of users is acting faithfully to the protocol. We propose an adaptive framework that based on the quality of service that each peer is receiving, will select a simpler protocol if the quality is good, or a stronger protocol that forces peers to contribute if the quality of service drops.

Acknowledgments We are grateful to Xavier Vilaça for the fruitful discussions and comments during the preparation of this report. This work was partially supported by Fundação para a Ciência e Tecnologia (FCT) via the INESC-ID multi-annual funding through the PIDDAC Program fund grant, under project PEst-OE/EEI/LA0021/2013, and via the project PEPITA (PTDC/EEI-SCR/2776/2012).

References

1. Liao, X., Jin, H., Liu, Y., Ni, L.M., Deng, D.: Anysee: Peer-to-peer live streaming. In: INFOCOM. Volume 25. (2006) 1–10
2. Bonald, T., Massoulié, L., Mathieu, F., Perino, D., Twigg, A.: Epidemic live streaming: optimal performance trade-offs. In: ACM SIGMETRICS Performance Evaluation Review. Volume 36., ACM (2008) 325–336
3. Jenkins, K., Hopkinson, K., Birman, K.: A gossip protocol for subgroup multicast. In: Distributed Computing Systems Workshop, 2001 International Conference on, IEEE (2001) 25–30
4. Stutzbach, D., Rejaie, R.: Understanding churn in peer-to-peer networks. In: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, ACM (2006) 189–202
5. Adar, E., Huberman, B.A.: Free riding on gnutella (2000)
6. Sirivianos, M., Park, J.H., Chen, R., Yang, X.: Free-riding in bittorrent networks with the large view exploit. In: IPTPS. (2007)
7. Locher, T., Moor, P., Schmid, S., Wattenhofer, R.: Free riding in bittorrent is cheap. In: Proc. Workshop on Hot Topics in Networks (HotNets), Citeseer (2006) 85–90
8. Feldman, M., Papadimitriou, C., Chuang, J., Stoica, I.: Free-riding and whitewashing in peer-to-peer systems. Selected Areas in Communications, IEEE Journal on **24**(5) (2006) 1010–1019
9. Feldman, M., Chuang, J.: Overcoming free-riding behavior in peer-to-peer systems. ACM SIGecom Exchanges **5**(4) (2005) 41–50
10. Typpi, T.: Game theory in peer-to-peer networks
11. Feldman, M., Lai, K., Stoica, I., Chuang, J.: Robust incentive techniques for peer-to-peer networks. In: Proceedings of the 5th ACM conference on Electronic commerce, ACM (2004) 102–111
12. Levin, D., Sherwood, R., Bhattacharjee, B.: Fair file swarming with fox. In: IPTPS. (2006)
13. Li, H.C., Clement, A., Wong, E.L., Napper, J., Roy, I., Alvisi, L., Dahlin, M.: Bar gossip. In: Proceedings of the 7th symposium on Operating systems design and implementation, USENIX Association (2006) 191–204

14. Li, H.C., Clement, A., Marchetti, M., Kapritsos, M., Robison, L., Alvisi, L., Dahlin, M.: Flightpath: Obedience vs. choice in cooperative services. In: OSDI. Volume 8. (2008) 355–368
15. Guerraoui, R., Huguenin, K., Kermarrec, A.M., Monod, M., Prusty, S.: Lifting: lightweight freerider-tracking in gossip. In: Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware, Springer-Verlag (2010) 313–333
16. Montresor, A., Jelasity, M.: Peersim: A scalable p2p simulator *
17. Agrawal, D., El Abbadi, A., Steinke, R.C.: Epidemic algorithms in replicated databases. In: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, ACM (1997) 161–172
18. Golle, P., Leyton-Brown, K., Mironov, I., Lillibridge, M.: Incentives for sharing in peer-to-peer networks. In: Electronic Commerce. Springer (2001) 75–87
19. Yang, B., Garcia-Molina, H.: Ppay: micropayments for peer-to-peer systems. In: Proceedings of the 10th ACM conference on Computer and communications security, ACM (2003) 300–310
20. Menasché, D.S., Massoulié, L., Towsley, D.: Reciprocity in peer-to-peer systems
21. Cohen, B.: Incentives build robustness in bittorrent. In: Workshop on Economics of Peer-to-Peer systems. Volume 6. (2003) 68–72
22. Gupta, M., Judge, P., Ammar, M.: A reputation system for peer-to-peer networks. In: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video, ACM (2003) 144–152
23. Karakaya, M., Korpeoglu, I., Ulusoy, O.: Free riding in peer-to-peer networks. Internet Computing, IEEE **13**(2) (2009) 92–98
24. Kermarrec, A.M., Massoulié, L., Ganesh, A.J.: Probabilistic reliable dissemination in large-scale systems. Parallel and Distributed Systems, IEEE Transactions on **14**(3) (2003) 248–258
25. Ganesh, A.J., marie Kermarrec, A., Massoulié, L.: Scamp: Peer-to-peer lightweight membership service for large-scale group communication, Springer-Verlag (2001) 44–55
26. Leitão, J., Pereira, J., Rodrigues, L.: Hyparview: A membership protocol for reliable gossip-based broadcast. In: In IEEE/IFIP International Conference on Dependable Systems and Networks, IEEE Computer Society (2007) 419–428
27. Li, B., Xie, S., Qu, Y., Keung, G.Y., Lin, C., Liu, J., Zhang, X.: Inside the new coolstreaming: Principles, measurements and performance implications. In: INFOCOM 2008. The 27th Conference on Computer Communications. IEEE, IEEE (2008)
28. Voulgaris, S., Jelasity, M., Van Steen, M.: A robust and scalable peer-to-peer gossiping protocol. In: Agents and Peer-to-Peer Computing. Springer (2005) 47–58