Efficient Free-rider Detection using Symmetric Overlays

(extended abstract of the MSc dissertation)

João Bruno Rodrigues Roque e Silva
Departamento de Engenharia Informática
Instituto Superior Técnico

Advisor: Professor Luís Rodrigues

***Abstract*—Edge-computing is one of the most promising techniques to leverage the excess capacity that exists at users' premises. Unfortunately, edge-computing may be vulnerable to free-riding, i.e., to nodes that attempt to benefit from the infrastructure without providing any service in return. In this paper we address free-riding in the context of edge-assisted streaming and propose the use of carefully crafted symmetric overlays to support message dissemination and efficient free-rider detection. The topology maintenance procedures of our overlay encourage nodes to maintain stable symmetric links. Leveraging the topological properties of the resulting symmetric overlay, simple and efficient tit-for-tat mechanisms allow to detect free riders without the signalling overhead of approaches that target at arbitrary topologies.**

I. Introduction

Decentralised peer-to-peer systems are a powerful tool to explore the unused capacity at the edges of the network. Unfortunately, it has been observed [1], [2], [3] that a significant fraction of nodes may free-ride by not contributing to the system, while still benefiting from the cooperation of a sufficiently large fraction of nodes that cooperate unconditionally, known as altruistic nodes. This not only puts an unfair load on altruistic nodes, but also degrades the performance of the tasks executed at the edge. Mechanisms that can detect free-riders and prevent them from dominating the system operation are therefore extremely relevant.

We address edge-assisted live-streaming. The usefulness of edge-computing to support live streaming has been demonstrated by several large scale real-life deployments, including PPLive[4] among others[5]. In this context, existing work [6], [7], [8] has addressed free-riding by assuming that nodes are rational. That is, nodes do not contribute because they aim at maximising a utility, which decreases with the amount of resources provided to other nodes. This assumption has two main drawbacks. First, because rational nodes can deviate from the protocol in an arbitrary fashion, sophisticated incentives are necessary, which are generally costly or require some degree of centralised control. Second, it is unreasonable to expect any node to be capable of calculating the optimal strategy that maximises its utility.

Therefore, we make the following assumption that, we believe, characterises most realistic settings. First, we assume that a significant fraction of nodes is altruistic. Second, we assume that the majority of nodes that deviate are free-riders. These nodes adopt the simple behaviour of never forwarding the stream. Third, we admit a small fraction of nodes that are rational and may adopt a more sophisticated behaviour. In this setting, it suffices to ensure that: i) free-riders are detected and expelled efficiently; ii) altruistic nodes have a way to detect that the assumptions have been violated (for instance, in the case where, against the expectations, there is a large fraction of rational nodes among the population), and can trigger a system adaptation to use more robust (but also more costly) incentive mechanisms; and iii) rational nodes still need to contribute to the system with a reasonable amount of resources in order to receive the stream.

Like most live-streaming implementations, we require nodes to join an overlay network in order to participate/receive the video broadcast. Instead of attempting to support arbitrary topologies, our overlay maintenance protocols include incentives for nodes to keep stable symmetric links with a small set of neighbours. As a result, it becomes possible to use simple tit-for-tat mechanisms to detect free-riders, instead of complex distributed monitoring and reputation mechanisms that may incur in a large signalling overhead. Based on these principles, we propose FastRank, an integrated topology management and peer-monitoring scheme that can effectively and efficiently minimize the impact of free-riders in live streaming applications. FastRank implements a ranking system that allows altruistic nodes to remain in the overlay and connect with other altruistic nodes, while free-riders are quickly penalised for not contributing to the system. Interestingly, we also show that the approach is robust to more sophisticated behaviour from rational nodes, such as attempts to manipulate the topology in their benefit, white-washing attacks, or contribution with just enough resources to avoid being marked as free-riders. In particular, we show that for these deviations to be profitable, the attackers still have to contribute with a reasonable amount of resources to the system. Furthermore, we show that, if the fraction of rational nodes is large, this can be detected by altruistic nodes that can trigger a system reconfiguration and commute to use more robust incentive mechanisms.

FastRank has been extensively evaluated. Our experimental results show that FastRank is able to keep altruistic nodes connected to each other, while isolating free-riders and denying them access to the stream. We also show that nodes adopting a more sophisticated behaviour receive a limited benefit and that, if their fraction is large, their presence can be detected by altruistic nodes.

The rest of the paper is structured as follows. Section II surveys related work. Section III presents the model. Section IV describes FastRank architecture and algorithms. Section VI reports on our experimental results. Finally, Section VII concludes the paper.

II. Related Work

There are two main approaches to support live-streaming in large scale systems: one is to build multicast dissemination trees and the other is to rely on gossip protocols. In optimal conditions, i.e. with minimal churn and a negligible number of free-riders, tree based approaches are a natural and efficient structure to spread information as they avoid any redundant message delivery. However, in an uncontrolled environment like the Internet, the performance of tree-vbased approaches degrades rapidly because tree need to be reconstructed whenever a node disconnects or does not conform with the protocol. Gossip protocol do not suffer from this limitation.

A gossip dissemination protocol can be decomposed in two components. The membership component is responsible for the definition and maintenance of the overlay topology. Challenges are in the definition of a topology that is simultaneously resilient, in order to mitigate the effects of churn, and efficient, presenting a small signalling cost even for a large number of participants. Two protocols that address this problem are SCAMP [9] and HyParView [10]. The second component of gossip is the dissemination strategy, which makes use of the overlay network to disseminate data. Message dissemination typically follows an epidemic style data dissemination, with nodes working on rounds, and selecting *fanout* nodes to relay data items in each round. This data distribution approach has shown to be effective and resilient to failures as long as non-failed nodes follow the protocol. However, the redundancy of gossip does not apply if nodes do not follow the protocol and avoid forwarding messages.

To mitigate the impact of nodes that do not follow the protocol, BAR Gossip [6] relies on a Balanced Exchange mechanism. In an interaction between two nodes n_1 and n_2, data relayed from n_1 to n_2 is encrypted before being sent and n_1 only releases the encryption key after receiving a comparable amount of data from n_2. The Balanced Exchange mechanism can be seen as a stronger implementation of the tit-for-tat approach that can be found in BitTorrent [2]. Although efficient, encryption imposes a considerable overhead, which is amplified in live content streaming due to the large number of packets exchanged and their timely delivery requirement.

FlightPath [7] contributes to attenuate the overhead of the brute-force approach proposed by BAR gossip in two ways. On one hand, it uses a flow control mechanism, where nodes negotiate in advance the exchange of packets. Flow control permits to balance evenly the traffic on each round and decrease redundancy. Secondly, FlightPath creates a balance account between pairs of nodes, which substitutes BAR Gossip's by a more efficient mechanism where some nodes may fall behind others although within previously agreed thresholds.

LiFTinG [8] implements a distributed mechanism for detecting rational nodes in asymmetric environments, i.e. when it is expected that a node sends more data to another node than it receives from him. LiFTinG requires nodes to track others nodes behaviour by cross-checking the history of their previous interactions. The randomness of node selection plays a key role in LiFTing by preventing colluding behaviour, where nodes can choose to send information always to the same subset of third parties, possibly uncovering a group's misbehaviour. However, cross-checking information incurs a non-negligible overhead.

III. System Model

We focus on the problem of disseminating a stream in a gossip fashion from a source to all the nodes of a peer-to-peer overlay network. The source first partitions the stream into multiple frames and sends them to a subset of nodes. Then nodes cooperate to disseminate the stream by forwarding each received frame to a subset of nodes selected at random. Each node n has an identifier and a *view* of the system, i.e., a subset of identifiers of other nodes to which n forwards frames. The nodes from the view of n are called the neighbors of n. The number of neighbours that n has in its view is the out-degree of n, and the number of neighbours that have n in their views is the in-degree of n. A view of n is symmetric if every node in that view has n in its view.

We assume that the source is trusted and never fails. On the other hand, nodes may not follow the protocol, either because they fail by crashing or because they gain from it. More precisely, we consider that nodes fall into one of three types: *atruistic*, *free-riders*, and *rational*. Altruistic nodes always follow the specified protocol. Free-riders always deviate from the protocol by never forwarding frames of the stream. Rational nodes strive to maximise a utility function, defined as the difference between the benefit of receiving the stream and the costs of executing the protocol, which comprise both communication costs and costs of performing computationally heavy computations. The benefit of receiving the stream is either (i) 0 if the reliability of the stream is not sufficiently high, or (ii) is a value larger than the costs of following the protocol. This definition embodies the assumption that rational nodes prefer to incur the costs of the protocol and receive the stream with minimum quality than to leave the system.

A strategy of node n can be deconstructed into (1) a *forwarding strategy*, specifying a probability distribution over the nodes from the view of n to which n forwards each previously received frame from the stream, (2) a *view strategy*, specifying the set of nodes that n keeps in its view, and (3) an *identity strategy*, specifying the set of different identities that n uses to participate in the system, in particular, n may perform White-washing and Sybil attacks by using many different identifiers [11], [12].

IV. FastRank

We now describe FastRank, a peer-to-peer streaming service that is resilient to free-riders and to a fraction of rational nodes. FastRank has three main components: an overlay network construction and maintenance protocol, a localised neighbour ranking mechanism, and a dissemination mechanism. These components cooperate in a synergetic manner to ensure that free-riders are promptly identified and shunned by their neighbors, so that they stop receiving the stream. In particular, the overlay maintenance mechanisms require nodes to preserve a small stable symmetric view. This, in turn, forces pairs of nodes to engage in long-term interactions called relationships. These interactions can be easily monitored locally, without requiring the dissemination of signalling traffic. Localised monitoring becomes then very efficient: nodes attribute a rank to each neighbour and expel from their view neighbours that have a low rank, i.e., which appear to be free-riders. Since rational nodes maximise their utility by avoiding being expelled by their neighbours, this encourages them to forward at least a fraction of the stream, ensuring a good streaming quality even in the presence of a significant fraction of rational nodes. Rational nodes may also attempt to keep a large view, to maximize the opportunities to receive new data. To prevent this, the overlay mechanisms make sure that the creation of new relationships incurs some utility loss. The same approach is used to prevent rational nodes from significantly increasing their utility by adopting new identifiers.

A. Overlay Network

The key idea behind FastRank is to leverage from the use of stable overlays with symmetric links to support the message dissemination. After establishing a link, two altruistic nodes will preserve it until one of them fails. The number of neighbours of each node is deliberately small (i.e, logarithmic with regard to the system size) [13], such that neighbours are required to interact frequently during the message dissemination process. As we will see, this allows to detect free-rider behaviour quickly and efficiently.

HyParView[10] is a peer-to-peer protocol that constructs and maintains an overlay with the properties required to implement our approach. Furthermore, the authors of HyParView have shown experimentally that their overlay could effectively support reliable muticast[14]. Unfortunately, we could not use HyParView as a black box while developping FastRank. In fact, HyParView appears to have been designed under the assumption that all nodes are altruistic. Moreover, it provides no support to prevent a node from constantly changing neighbours. Given that nodes will require some time to identify a new neighbour as a free-rider, the free-rider will receive some packets before it is disconnected. If the free-rider can create new links at the same pace it looses old ones, it will still be able to receive the stream. Therefore, FastRank implements a variant of HyParView. This adaptation of HyParView, that we call *Constrained HyParView*, implements mechanisms that constrain the pace at which a node can establish new relationships.

1) Constrained HyParView: Constrained HyParView is a redesign of the original HyParView protocol that we have developed to meet the requirements of FastRank. This variant, includes mechanisms that constrain the rate at which a node can establish new relationships in the overlay. For this purpose, when a node i contacts another node j, to create a new relationship between i and j, node i is given a time-consuming task that it needs to perform before the relationship request is accepted. This, in practice, introduces a *quarantine period* before the establishment of a new relationship. The quarantine period should be long enough such that multiple frames are lost and white washing becomes unappealing. Furthermore, the task given to node i should be such that more than one task cannot be performed in parallel by a singe node during a given quarantine period, i.e., if a node attempts to establish two new relationships, it should be forced to "pay" the cost of waiting two quarantine periods.

To achieve the goals above, in FastRank, we have opted to use crypto-puzzles. More precisely, when a node i contacts node j to establish a new relationship, node j prepares a crypto-puzzle that needs to be solved by i in order for j to accept the relationship. The crypto-puzzle is such that the estimated average time to solve it is the pre-defined quarantine period, even if the entire computing resources of a node are devoted to the task. Several examples of crypto-puzzles with these guarantees have been described in the literature[15], [16], [17]; in FastRank we have opted to use [18].

A node joining the Constrained HyParView overlay contacts a target number of neighbours, gets a challenge from them, solves the crypto-puzzles and provides the answers to all neighbours simultaneously. In this way, it is likely that the joining node is accepted by a number of neighbours that is large enough to initiate its operation without risking being marked as free-riders (because it does not receive enough information to forward). A significant advantage of this approach is that a node attempting to join the network is constrained by its own resources, no matter how many different nodes it tries to contact or how many identities it attempts to use, since the number of tasks it may perform by unit of time is limited by the finite hardware resources of the node. Therefore, FastRank also mitigates the impact of White-washing and Sybil attacks.

The reader should notice that the mechanism above is asymmetric: while the node that attempts to establish a relationship has to solve the crypto-puzzle, the node accepting the relationship does not. The reason for this is that FastRank is designed to detect, and isolate, free-riders in the overlay. Rational nodes that are isolated will likely attempt to join again, by contacting different nodes. We aim at minimising the negative effect that this behaviour has on altruistic nodes, while still allowing new altruistic nodes to join the overlay at any moment. Also, if a node receives multiple relationship requests concurrently, it will submit a different crypto-puzzle to each node attempting to establish a relationship and then it will only connect to the first node to complete the task. Thus, if a free-rider attempts to solve multiple crypto-puzzles at the same time, and competes with altruistic nodes for relationships, it may risk not to be accepted, given that nodes that devote all resources to solving a single crypto-puzzle are more likely to respond first and obtain the relationship. Furthermore, a node is forced to engage in repeated interactions with its neighbours immediately after the relationship is established. A free-rider, that will only consume frames from a relationship will be detected and see the relationship ended before it is able to acquire a new relationship.

In HyParView a node pro-actively attempts to maintain his active view full. Therefore, if a neighbour crashes, it immediately attempts to establish a new relationship to refill its active view. However, in Constrained HyParView, actively attempting to establish a new relationship is costly. Furthermore, if very few nodes have empty slots in their active views, it is likely that multiple nodes concurrently compete for that entry (and only one will succeed). This further exacerbates the cost of joining an overlay where all nodes pro-actively attempt to fill their views as soon as possible. On the other hand, an altruistic node can opt to wait and refill its active view by accepting relationships from nodes attempting to join the network. If all altruistic nodes do this, not only they avoid the costs of initiating relationships but also they make the joining procedure easier for new altruistic nodes that want to be part of the overlay. In Constrained HyParView we use a low watermark threshold, denoted *baseview*, that needs to be reached before the node pro-actively looks for neighbours. If some relationships end but the size of the active view is above *baseview*, the node simply waits for join requests, and will accept relationships until *maxview* is reached. This is a safety mechanism that allows nodes to ensure that newcomers have the possibility to join the system. When a node n has in its view *maxview* relationships and n receives a join request, n will request the new node to solve a crypto-puzzle. The first node to complete it will replace the node with the lowest rank in its view.

Finally, in FastRank, the source of the stream is treated differently from every other node. The source does not keep an explicit active view to a fixed set of nodes. Instead, it uses a large passive view to select a number of contact points at random for each frame it sends. Nodes always receive frames sent directly by the source and altruistic nodes forward the frames to the neighbours in their active view (a detailed description is provided below). The goal is to distribute the load evenly among the members of the overlay, such that there is not a fixed set of nodes that is close to the source (and always has to forward frames) and another set of nodes that permanently act as leafs (and just receive frames).

B. Ranking Algorithm

FastRank leverages from the topological properties of Constrained HyParView to implement an efficient localised monitoring mechanism that can effectively detect and, ultimately, expel free-riders from the active view of nodes. The mechanism is based on the observation that, in steady relationships, two altruistic nodes roughly send the same amount of frames to each other. If the balance of exchanged frames is highly asymmetric, this is a sign that the node that is receiving but not forwarding frames is likely a free-rider or a failed node, and thus may be expelled from the view. The fact that Constrained HyParView keeps symmetric views plays a crucial role in this mechanism, since it ensures that altruistic nodes have a small set of neighbours with whom they interact repeatedly. This allows to detect any unbalance quickly.

The balance of the exchanges with a neighbour is captured by FastRank as a numeric rank[19]. Each node i maintains, for each neighbour j in its active view a separate rank_{ij}. The rank of a new neighbour is initiated to a predefined value, denoted the *baserank* and then maintained using a very simple rule that consists of incrementing the rank of the neighbour by one unit every time a frame is received from that neighbour and by decrementing the rank also by one unit when a frame is sent to the neighbour. In FastRank, the value of *baserank* is simply 0. This means that a neighbour that sends more frames than it receives keeps a positive score, and a free-rider will have negative score. Furthermore, the ranking algorithm also defines a minimum threshold for the rank, denoted *minrank*, below which a node is expelled from the view. In Section VI, we discuss how an appropriate value for *minrank* can be selected.

C. Dissemination Mechanism

Frame dissemination is implemented as follows. The stream source selects, for each frame, a number f of contact points among the entire set of members of the overlay (this is achieved by keeping a large passive view) and then sends the frames to those nodes. When a node receives a frame, directly from the source or from one of its neighbours, first checks if the frame is a duplicate (for this purpose, each node keeps a record with the identifiers of the last frames it has received). Duplicate frames are simply discarded and never forwarded to any neighbour.

If the frame is new, the node will forward the frame to each of its neighbours with a probability that is a function of the rank of that neighbour. The size of the active views of the overlay constructed by the Constrained HyParView are deliberately small but still large enough to tolerate a large fraction of faulty nodes. As a result, if all nodes are altruistic, the use of flooding in the overlay may generate many redundant messages. Therefore, an altruistic node forwards a message to other altruistic nodes with a probability lower than 1 that is called the *base forwarding probability* (or simply *bfp*). The value of *bfp* is selected such that the reliability of the dissemination is still ensured but with a much smaller cost than that incurred when flooding is used. Naturally, the value of the *bfp* depends on the size of the active view. In Section VI we discuss how *bfp* can be configured for optimal results.

Notice that, prior to forwarding any frame, all nodes in the active view have a rank above *minrank*. Frames are forwarded to nodes that have a rank above *baserank* with the probability *bfp*, and are forwarded with a probability lower than *bfp* but still larger than 0 for the members of the view whose rank is above *minrank*. More precisely, FastRank uses the following formula to compute the forwarding probability from node i to a neighbour j, denoted $\mathit{fp}_i(j)$:

$$\mathit{fp}_i(j) = \begin{cases} \mathit{bfp} & \text{if } \mathit{rank}_{ij} \geq \mathit{baserank} \\ \mathit{bfp}\left|\frac{\mathit{minrank}-\mathit{rank}_{ij}}{\mathit{minrank}}\right| & \text{if } \mathit{baserank} > \mathit{rank}_{ij} \geq \mathit{minrank} \end{cases}$$

This guarantees that the forwarding probability to neighbours with a rank lower than *minrank* decreases with the rank, thus decreasing the expected benefit of rational nodes that forward frames with probabilities lower than what is specified by the protocol. Such decrease persuades rational nodes to forward frames with a probability strictly higher than the smallest possible probability required for keeping its rank in other neighbours above *minrank*.

V. Considered Behaviours

FastRank has been designed for systems where most of the nodes are altruistic and only a small fraction of nodes are free riders or rational. The case where all nodes are rational is outside the scope of this paper. As discussed in the related work section, this scenario requires considerably more expensive protocols. In the evaluation section we analyse the performance of the system for different fractions of nodes in the population that deviate from the protocol (misbehaving nodes), considering that nodes can belong to one of the following categories.

- *Altruistic nodes:* An altruistic node follows the specified protocol without ever deviating from the specification.

- *Free-Riders:* Are nodes that never forward packets. They receive information from neighbours until they are expelled. Furthermore, they try to maximize the number of neighbours they have.

- *Rational Nodes:* Nodes that attempt to deviate in order to maximize their utility. We discuss the behaviour of rational nodes below.

A. Rational Behaviour

We consider that a rational node obtains a benefit every time it receives a new packet and incurs in a cost every time it sends a packet or solves a crypto-puzzle. Thus, to maximize its utility, a rational node aims to increase the ratio between received and sent packets. It is clear from the description of the algorithm that, if a node does not keep a balanced exchanged of information with a neighbour, it is eventually detected as a free-rider and expelled by that neighbour. There are then two main possible strategies that a rational node may follow to increase its utility:

- *Shrinking the active view:* A rational node may opt to reduce the size of its active view, such that it receives the stream but has to forward packets to as few neighbours as possible. Still, it behaves as an altruistic node to the nodes that it keeps as its neighbours. Thus, when deviating in this way, the node appears to be altruistic to its neighbours, that have no way to detect that it is using a reduced fanout. Note that more sophisticated algorithms (e.g. LiFTinG [8]), that do not rely exclusively on localised information, have been designed to detect this attack.

- *Slowdown (maintaining the score strictly above the watermark level):* A rational node may also reduce the rate at which it forwards packets in such a way that it is able to maintain its score above *minrank* and, therefore, never be expelled from the active view of altruistic nodes. Furthermore, it may try to get as much neighbours as possible. The ideia is that if it keeps enough neighbours, it may still receive the stream with good quality, despite forwarding messages only seldom.

We will show that, in face of the deviations above, FastRank exhibits the following interesting properties: i) the quality experienced by altruistic nodes is only mildly affected, even for large fractions of misbehaving nodes; ii) rational nodes still have to contribute with a reasonable amount of resources to the system in order to get the stream with a minimal quality. As a result, the unbalance between the resources committed by altruistic nodes and misbehaving nodes is not large (10% less, at most); iii) if a large fraction of misbehaving nodes exist among the population, this can be detected by the altruistic nodes.

VI. Evaluation

In this section, we provide an extensive evaluation of FastRank. All experiments were performed using the PeerSim Framework [20]. Simulations used 1000 nodes and consisted in the dissemination of 20000 frames, each injected in the network by the streamer using 7 peers randomly chosen. Results presented in this section are the average of 100 independent runs using the same configuration.

The evaluation is divided into four different parts. In the first part, we support the choice of values selected for the different parameters of the system. The second part illustrates the operation of the system when all nodes are altruistic. The third part discusses the effects of free-riders and the fourth part the effect of rational nodes in the system.

A. Configuring FastRank

The parameters that affect the operation of FastRank are the following: the size of the active view and of *minview*; the parameter of the ranking procedure (*minrank*); the base forward probability *bfp* used in the dissemination process; and the average length of the *quarantine period* (i.e., the average time needed to solve the crypto-puzzle when creating a new relationship). Table I presents the default configuration values of FastRank. We discuss the rationale for configuration of these different parameters in the following subsections. Unless stated otherwise, the discussion applies to a network of 1000 nodes (the setup that has been used for the graphs depicted in the paper).

	value
maxview	15
baseview	12
minrank	-15
bfp	0.4
quarantine period (frames)	220

Table I
DEFAULT CONFIGURATION OF FASTRANK

1) View Size: We first discuss how the size of the active view is selected. For now, lets assume that flooding is used to propagate the messages in the overlay (in the next section, we discuss why flooding is not used in FastRank). As long as the network of altruistic nodes remains connected, all correct nodes will receive all the messages. Therefore, the *baseview* size must be selected such that the likehood of an altruistic node to become isolated in the presence of faulty nodes is very small. We have opted to configure FastRank such that at least 30% of faulty nodes can be tolerated with minimum effect on the altruistic nodes. Figure 1(a) shows the percentage of altruistic nodes that becomes isolated from the primary components of the overlay (the primary component is the largest connected subgraph in the overlay) for different sizes of the active view, after 30% of simultaneous failures. As it can be seen, if the *baseview* is equal or larger than 11, only 0.1% of altruistic nodes are isolated. Such a small value motivated us for using the conservative approach of selecting 12 as the default value for *baseview*. We have opted to add 3 additional slots to facilitate the inclusion of joining nodes, for a *maxview* size of 15.

2) Forward Probability: We now explain the rationale for selecting the message base fowarding probability *bfp*. In the spirit of gossip protocols, we avoid this redundancy by forwarding messages with a given probability smaller than 1. By fixing the *baseview* to 12, Figure 1(b) depicts the reliability of the streaming protocol on a FastRank overlay, as a function of the dissemination probability and Figure 1(c) depicts the resulting redundancy. As it can be observed, by selecting a forward probability of 0.4 on an overlay where the*baseview* is 12, one can still achieve a very high reliability with a significant reduction in the redundancy of the dissemination procedure.

3) Rank Maintenance: The goal of the rank mechanism is to detect free-riders by equating a rank below *minrank* with free-riding behaviour. However, due to random fluctuations of dissemination, it is possible that the rank of altruistic nodes also drops below *minrank*, a situation that we call a *false positive*. Our goal is then to promptly detect free-riders while minimising false positives. Therefore, the value of *minrank* should weigh this trade-off. Figure 2(b) shows the fluctuation of the rank among two altruistic nodes. From these graphs it is clear that *minrank* should not be higher than -15. However, note that the lower the value of *minrank*, the less likely it is to generate a false positive but also the longer it would take to detect a free-rider, as shown in Figure 2(c). Since we aim at a fast detection of free riders, we have opetd to use the maximum value that ensures a small faction of false positives, i.e, the value of -15.

4) Quarantine Time: As discussed before, the goal of the quarantine time is to ensure that nodes cannot replace relationships faster than they are ended. From Figure 2(c) it can be observed that the last free-rider was detected after 110 frames for a *minrank* of -15. Therefore, the join procedure should use a crypto-puzzle that takes, on average, a time that is longer than the time it takes to forward that number of frames. Since we also aim at penalizing free-riders, we have selected a quarentine period of twice the detction time (i.e, corresponding to 220 frames). In this way, free-riders that continuously attempt to replace old neighbours by new neighbours miss 50% of the frames.

B. Failure-free Operation

In this subsection, we provide some additional insights on the operation of FastRank in a failure-free scenario, i.e., in a setting where all nodes are altruistic and do not fail. For this, we consider a stream of 24 frames per second, generated by one single stream source, during a complete session with 14 minutes. The session starts with 1000 nodes and in the middle of the stream (at minute 7), 100 additional nodes join the stream (i.e., at that momento we induce an 10% increase in the overlay population). With this setting, we show: the time it takes to setup the FastRank overlay, the reliability experienced by a node; the average number of retransmissions per frame received during the streaming session; the percentage of false-positives during the session; the amount of crypto-puzzles solve by each node during the session; and, finally, the time it takes for a new node to join an ongoing streaming session with 90% reliability. The results are depicted in Table II.

Initial network size	1000
Joining nodes	100
False positives	0.03
Global reliability	0.999
Newcomers reliability	0.995
Newcomers average puzzles	12.2
Worst case time to join stream (frames)	3116

Table II
OPERATION IN ABSENCE OF MISBEHAVIOUR

As it can be seen, the amount of cripto puzzles solved by joining members, 12.2, is slightly above the minimum (we note that a joining members must establish *baseview*, i.e., 12, neighbours). The difference is due to the contention for the free slots in the views of nodes that already belong to the overlay. But even in the worst case (for the last node to join) the time corresponds to solving 14 crypto puzzles, just 2 above the minimum. This contention is minimal and shows that having a *maxview* above *baseview* is quite effective at helping new members to join the overlay. It can also be observed that the joining of new members does not affect the reliability of the stream nor the accuracy of the free-rider detection mechanism.

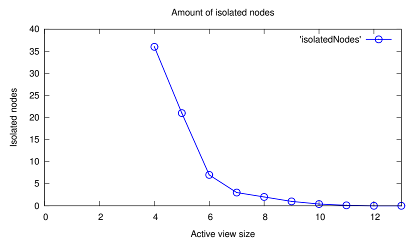
C. Tolerating Free-Riders

In this section, we provide some additional insights on the operation of FastRank in the presence of free-riders. In this experiments we let the system run with 100% altruistic nodes until a point where a fraction of all nodes adopts the behaviour of a free-rider. Figure 3 shows the evolution of the composition of the active views of nodes after the fault is injected. The figure shows that after 110 frames, free-riders are detected and the system starts to reconfigure. After 2500 frames the system stabilizes in a configuration where 95% of the members in the active view of an altruistic node are other altruistic nodes. Consequently, free-riders have become disconnected from the network.

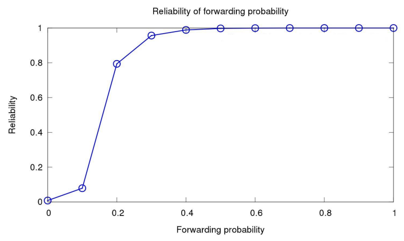
We have also measured how many crypto-puzzles altruistic nodes and free-riders have solved during the reconfiguration, for 30% free-riders in the system. Since the *baseview* is 12, with 30% free-riders, after the attack an altruistic node must, on average, replace 4 members in its view. However, in this experiment, an altruistic node has to solve 6.75 crypto-puzzles before it stabilizes its active view with other altruistic nodes, i.e., almots 3 more puzzles than in the ideal case. This is due to the fact that this experiments captures an extreme case, where all free-riders act simultaneously, and also keep continuously solving crypto-puzzles to replace their broken relationships, thus generating a significant contention for the free slots in the view of altruitic nodes.

D. Tolerating Rational Nodes

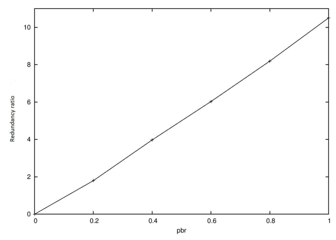
Given that rational nodes only obtain a benefit for receiving the stream with sufficiently high reliability, this shows that they do not have incentives to free-ride. In this subsection, we provide some additional insights on the operation of FastRank when a fraction of nodes is rational and deviates following strategies that maximise their utility. Recall that the utility is the difference between the benefits and the costs of executing the protocol. We consider that the only significant costs of



(a) Isolated nodes after 30% failures.

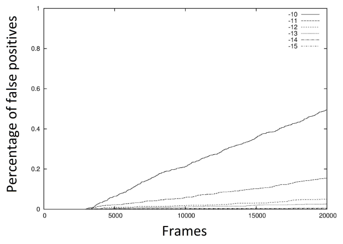


(b) Reliability.

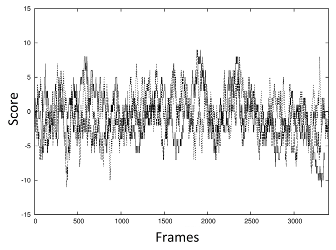


(c) Redundancy.

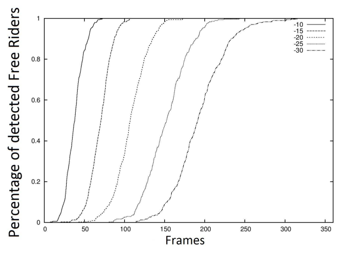
Figure 1. Baseview and Forward Probability (reliability vs redundancy)



(a) False positives.



(b) Rank fluctuation.



(c) Detection Time.

Figure 2. Ranking

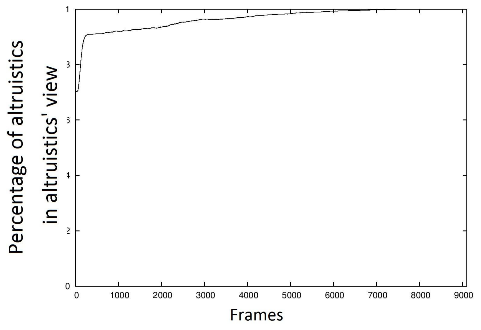


Figure 3. Recovering from free-riders

our protocol are those of forwarding frames and computing crypto-puzzles. We also assume that the fraction of rational nodes is constant.

1) Strategies: Recall also a strategy of a rational node can be decomposed into the forwarding, view, and identity strategy. We fix the identity strategy to one identity per node. Later, we discuss the effect of strategies that use multiple identities per node. Regarding the forwarding strategy, nodes can adopt a *free-riding* strategy by not forwarding anything, may follow an *altruistic* strategy by following the protocol, or may follow a *minimum forwarding* strategy, by forwarding the minimum number of frames required to avoid losing relationships. This covers all relevant possibilities. First, forwarding frames to more nodes on average only increases the cost. Second, a rational node does not benefit from forwarding a number of frames lower than the minimum required to keep relationships but not to free-ride. Finally, the minimum forwarding strategy models the worst-case scenario where the utility decrease due to nodes not maintaining a rank above *baserank* does not dissuade them from deviating. Regarding the view, the only possibilities are the *enlarged view*, the *shrunk view*, and the *same view* strategies.

We have seen that the same view or shrunk view strategies in combination with a free-riding strategy should not provide any gain, since the reliability of the stream drops significantly. Moreover, an altruistic strategy combined with an enlarged view increases the costs of performing crypto-puzzles without increasing the benefit, since the reliability of the stream remains roughly the same. Hence, we can focus on the following five deviations: free-riding with enlarged view, altruistic with shrunk view, and minimal forwarding with same, shrunk, and enlarged views.

We evaluate these strategies according to two main criteria: (1) impact on the reliability of rational nodes and (2) impact on the reliability of altruistic nodes when a fraction of rational nodes deviate. We show that, whenever rational nodes may increase their utility, FastRank exhibits the following interesting properties: (i) the reliability experienced by altruistic nodes is only mildly affected, even for large fractions of rational nodes; (ii) rational nodes still have to contribute with a reasonable amount of resources to the system in order to get the stream with a minimal reliability; as a result, the unbalance between the resources committed by altruistic nodes and rational nodes is not large (10% less, at most); and (iii) if the fraction of rational nodes is large, then this can be detected by the

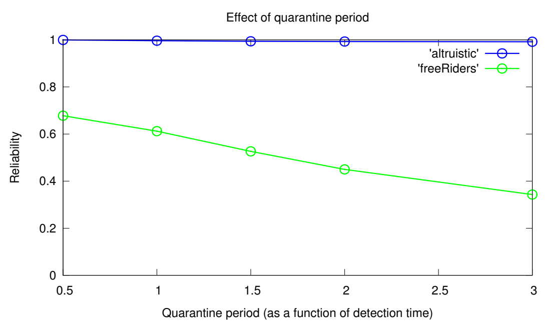


Figure 4. Effect of quarantine period

Rational nodes' active view size	**-**	**3**	**4**	**5**	**6**
Max hop ratio	1	1.27	1.45	1.37	1.09
Avg hop	1	1.07	1.08	1.13	1.11
Altruistic reliability	0.99	0.99	0.99	0.99	0.99
Rational reliability	-	0.03	0.06	0.67	0.88

Table III
EFFECT OF ALTRUISTIC WITH SHRUNK VIEW STRATEGY (30% OF RATIONAL NODES)

altruistic nodes.

2) Free-riding with Enlarged View Strategy: The main gain of this strategy may stem from avoiding the costs of forwarding frames while avoid becoming isolated. This may be an optimal strategy if the cost of executing crypto-puzzles is lower than that of forwarding frames and the quarantine period is not sufficient to prevent the node from receiving the stream with minimum reliability. Figure 4 shows the reliability experienced by a rational node adopting this behaviour. Knowing that, while on quarantine, a node is unable to search for another relationship, the figure depicts reliability values for different quarantine times. It can be observed that if the quarantine time is larger than the detection time, then the proportion of frames received by the free-riders, drops significantly. This shows that, with the right choice of the quarantine time, rational nodes do not increase their utility by adopting this behaviour.

3) Altruistic with Shrunk View Strategy: This strategy is more beneficial to rational nodes when the cost of computing crypto-puzzles dominates communication costs, such that nodes aim at minimising the number of relationships, while still keeping the streaming reliability to a minimum. In Table III, we show the composition of the views of both altruistic and rational nodes, the reliability of the stream after the deviation, and the increase in message latency caused by the deviation. Since rational nodes consistently end a fraction of their relationships, altruistic nodes tend to replace those relationships. Therefore, the network remains connected and the reliability is not significantly affected by the deviation. On the other hand, since a fraction of nodes has a smaller out-degree, the latency of the dissemination increases. With a sufficiently large fraction of rational nodes, the increase in the latency is noticeable. This open the door for an adaptive solution, where we can trigger a change from FastRank to a more costly protocol that detects and punishes rational behaviour (e.g. LiFTinG [8]).

Reliability of altruistic nodes after the deviation	0.97
Reliability of rational nodes after the deviation	0.50
Fraction of frames sent by rational nodes	0.2

Table IV
EFFECT OF MINIMAL FORWARDING WITH SAME VIEW STRATEGY (30% OF RATIONAL NODES)

Rational active view size	**12**	**20**	**25**	**30**	**35**	**40**
Ratio of solved crypto puzzles	1	19	22	42	45	49
Average frame ratio	0.24	0.40	0.50	0.59	0.66	0.80
Reliability of rational	0.71	0.80	0.86	0.98	0.99	0.99

Table V
EFFECT OF MINIMAL FORWARDING WITH ENLARGED VIEW STRATEGY (FOR ONE RATIONAL NODE)

4) Minimal Forwarding with Same and Shrunk View Strategies: The advantage of these strategies is that if a node keeps enough relationships, it may still receive the stream with sufficiently high reliability, despite forwarding frames only seldom. Table IV show the effect of the minimal forwarding with same view strategy. As it can be seen, it is severely penalized, as its perceived reliability drops significantly. On the other hand, even for 30% of rational nodes, this deviation has no negative impact on the reliability experienced by altruistic nodes. Since the reliability only drops by shrinking the view, rational nodes clearly do not gain from following the minimal forwarding with shrunk view strategy, so we opt to omit the evaluation of this strategy.

5) Minimal Forwarding with Enlarged View Strategy: The previous results showed that a minimal forwarding strategy is harmful due to a significant decrease in the reliability. A rational node may circumvent this problem by enlarging the view. Table IV shows how long it takes for such a node to receive the stream with the same reliability of an altruistic node. It can be observed that a node with a score of *minrank* on all of its incoming links, needs to have a constant active view of size at least 25 to approximate the reliability of an altruistic node. Thus, it need to solve 22 times more crypto-puzzles than altruistic nodes to achieve that state. Furthermore, since it needs to keep all these relationships, it still needs to forward frames, but approximately 80% of that of an altruistic node. Therefore, this strategy is less profitable than altruistic with shrunk view, because with this deviation a rational nodes has the same forwarding effort than a node that shrinks its view, with the penalty of performing more crypto-puzzles.

E. White-washing and Sybil Attacks

To obtain a relationship, a node must solve a crypto-puzzle, which takes more time than the time it takes to detect it as a free rider. Thus, a node cannot replace relationships fast enough to sustain the reception of the stream with sufficiently high quality. This mechanism is completely independent of the identity a node opts to present to its neighbours. Also, nodes do not maintain any record of past interactions based on identifiers, and a node that fails to maintain a relationship active is punished with a crypto-puzzle every time it attempts to replace that relationship, regardless of the identity it opts to present. Given this, a strategy where a node uses multiple identifiers is equivalent to enlarging the view. This implies that

White-washing and Sybil attacks are no more effective than strategies where a node employs a single identifier and keeps an enlarged view.

VII. Conclusions

In this paper we have presented FastRank, a peer-to-peer streaming protocol that relies on an overlay network with symmetric links to mitigate the effect of free riders in an efficient and effective manner. FastRank includes overlay construction mechanisms that encourage nodes to perform repeated interactions with a small number of nodes and then leverages from this property to implement efficient localised scoring mechanisms that can be used to detect and expel free riders. The resulting system can tolerate up to 30% of free riders without decreasing the reliability of the stream. As a result, it allows for an optimised operation in the case where free riders or rational nodes are residual (which happens often in practice). FastRank is in contrast with more robust solutions, that tolerate wider range of attacks at a much higher cost. Interestingly, FastRank can also tolerate a fraction of more sophisticated rational behaviour, with small but detectable impact on the perceived streaming process. This open the door for adaptive solutions, where FastRank is used while the number of misbehaving nodes is small, and the operation reverts to more expensive solutions such as Lifting when the number or the sophistication of attackers increases.

Acknowledgments

This work was partially supported by Fundação para a Ciência e Tecnologia (FCT) via the INESC-ID funding with reference UID/CEC/50021/2013 and via the project PEPITA (PTDC/EEI-SCR/2776/2012). Parts of this work have been performed in collaboration with other members of the Distributed Systems Group at INESC-ID, namely, Prof. Hugo Miranda and Xavier Vilaça.

References

[1] E. Adar and B. Huberman, "Free riding on gnutella," *First Monday*, vol. 5, no. 10, Oct. 2000. [Online]. Available: http://firstmonday.org/issues/issue5˙10/adar/index.html

[2] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, 2003, pp. 68–72.

[3] D. Hughes, G. Coulson, and J. Walkerdine, "Free riding on gnutella revisited: The bell tolls?" *IEEE Distributed Systems Online*, vol. 6, no. 6, pp. 1–, Jun. 2005.

[4] Y. Huang, T. Fu, D.-M. Chiu, J. Lui, and C. Huang, "Challenges, design and analysis of a large-scale p2p-vod system," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 375–388, 2008.

[5] X. Liao, H. Jin, Y. Liu, L. Ni, and D. Deng, "Anysee: Peer-to-peer live streaming." in *INFOCOM*, vol. 25, 2006, pp. 1–10.

[6] H. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin, "Bar gossip," in *OSDI*, 2006, pp. 191–204.

[7] H. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robison, L. Alvisi, and M. Dahlin, "Flightpath: Obedience vs. choice in cooperative services." in *OSDI*, vol. 8, 2008, pp. 355–368.

[8] R. Guerraoui, K. Huguenin, A.-M. Kermarrec, M. Monod, and S. Prusty, "Lifting: lightweight freerider-tracking in gossip," in *Middleware*, 2010, pp. 313–333.

[9] A. Ganesh, A.-M. Kermarrec, and L. Massouli, "Scamp: Peer-to-peer lightweight membership service for large-scale group communication." Springer-Verlag, 2001, pp. 44–55.

[10] J. Leitão, J. Pereira, and L. Rodrigues, "Hyparview: A membership protocol for reliable gossip-based broadcast," in *IEEE DSN*, 2007, pp. 419–428.

[11] J. Douceur, "The sybil attack," in *Peer-to-peer Systems*. Springer, 2002, pp. 251–260.

[12] P. Resnick *et al.*, "The social cost of cheap pseudonyms," *Journal of Economics & Management Strategy*, vol. 10, no. 2, pp. 173–199, 2001.

[13] A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh, "Probabilistic reliable dissemination in large-scale systems," *IEEE Trans. on Parallel and Distributed Systems,*, vol. 14, no. 3, pp. 248–258, 2003.

[14] J. Leitão, J. Pereira, and L. Rodrigues, "Epidemic broadcast trees," in *IEEE SRDS*, Beijing, China, Oct. 2007, pp. 301–310.

[15] X. Wang and M. Reiter, "Defending against denial-of-service attacks with puzzle auctions," in *Security and Privacy*. IEEE, 2003, pp. 78–92.

[16] R. Merkle, "Secure communications over insecure channels," *Comm. of the ACM*, vol. 21, no. 4, pp. 294–299, 1978.

[17] N. Borisov, "Computational puzzles as sybil defenses," in *IEEE P2P 2006*, 2006, pp. 171–176.

[18] W.-C. Feng, E. Kaiser, and A. Luu, "Design and implementation of network puzzles," in *IEEE INFOCOM 2005*, vol. 4, 2005, pp. 2372–2382.

[19] M. Karakaya, I. Korpeoglu, and O. Ulusoy, "Free riding in peer-to-peer networks," *Internet Computing, IEEE*, vol. 13, no. 2, pp. 92–98, 2009.

[20] A. Montresor and M. Jelasity, "Peersim: A scalable p2p simulator *."