Efficient Free-rider Detection using Symmetric Overlays

João Bruno Rodrigues Roque e Silva

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Luís Eduardo Teixeira Rodrigues

Examination Committee

Chairperson: Prof. Daniel Jorge Viegas Gonçalves

Supervisor: Prof. Luís Eduardo Teixeira Rodrigues

Member of the Committee: Prof. Fernando Manuel Valente Ramos

October 2015

Acknowledgements

I would like to thank my advisor, Professor Luís Rodrigues for giving me the opportunity to work on this thesis under his supervision. His advices and motivation were a fundamental key to the development of this thesis. Moreover, I would like to thank Prof. Hugo Miranda and Xavier Vilaça for all of their valuable suggestions and advices in order to make my thesis a success.

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, via the project PEPITA (PTDC/EEI-SCR/2776/2012).

Lisboa, October 2015

For my family,

Resumo

Hoje existe uma grande capacidade computacional na periferia da rede, que pode ser usada para aumentar a capacidade de escala dos sistemas. Infelizmente, ao usar componentes que são administrados por utilizadores finais, o sistema fica sujeito a comportamentos de parasitagem, em que alguns nós tentam beneficiar do serviço sem disponibilizar os seus recursos. Neste artigo estudamos técnicas que permitem evitar comportamentos parasitas em sistemas de difusão entre-pares. Para este efeito, desenvolvemos uma variante de uma rede sobreposta em que todos os nós possuem vistas parciais simétricas, de tamanho semelhante. Estas vistas são usadas para promover interacções frequentes entre vizinhos, que facilitam a monitorização e a classificação dos parceiros de forma localizada e eficiente. Esta classificação é usada para aplicar sistemas de penalização simples, que permitem detectar e excluir os nós parasitas em sistemas de transmissão contínua de dados (streaming). A avaliação mostra que este mecanismo limita a assimetria na utilização de recursos mesmo na presença de comportamentos mais sofisticados.

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.

Palavras Chave
Keywords

Palavras Chave

Disseminação de informação

Comportamento parasita

Comportamento racional

Fiabilidade

Arquitectura entre pares

Keywords

Information dissemination

Free Rider behavior

Rational behavior

Reliability

Peer to peer architecture

Contents

List of Figures

List of Tables

1 Introduction

The task of disseminating, in real-time, multimedia content to a large number of users has been coined *live-streaming*(Liao, Jin, Liu, Ni, and Deng 2006; Bonald, Massoulié, Mathieu, Perino, and Twigg 2008). 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(Jenkins, Hopkinson, and Birman 2001) among peers have emerged as a promising strategy, due to their scalability and robustness to failures and churn(Stutzbach and Rejaie 2006).

1.1 Motivation

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 (Adar and Huberman 2000a; Cohen 2003; Hughes, Coulson, and Walkerdine 2005) 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.

In this work 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(Huang, Fu, Chiu, Lui, and Huang 2008) among others(Liao, Jin, Liu, Ni, and Deng 2006). In this context, existing work (Li, Clement, Wong, Napper, Roy, Alvisi, and Dahlin 2006; Li, Clement, Marchetti, Kapritsos, Robison, Alvisi, and Dahlin 2008; Guerraoui, Huguenin, Kermarrec, Monod, and Prusty 2010) 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.

1.2 Contributions

This work addresses the problem of selfish users in a live streaming paradigm under a peer-to-peer architecture. To achieve the goals above, the thesis contributes with

- A new algorithm that is an integrated topology management and peer-monitoring scheme that can effectively and efficiently minimize the impact of free-riders in live streaming applications.
- An extensive experimental evaluation of the proposed algorithm that provides insights on its advantages and limitations.

1.3 Results

The results of this work can be enumerated as follows:

- A specification of the algorithm.
- An implementation using the PeerSim Framework.
- An extensive experimental evaluation using the resulting prototype.

1.4 Research History

The aim of our work was to explore the different behaviors a user can adopt in a peer-to-peer topology and provide a fast and low-cost algorithm when the population is willing to contribute for the information dissemination and provide the necessary mechanisms to adopt a different protocol when the population is trying to benefit from the system without providing the expected contributions. To address the first part of this problem, we created a low overhead novel algorithm based on HyParView. This algorithm has the best results when the entire population follows the specified protocol or when selfish users are free riders. Since users can deviate from the protocol in arbitrary ways, we rely on LiFTinG, a more costly and heavier protocol, to address this limitation. In my work I have benefited from the collaboration of my advisor, Prof. Luís Rodrigues, Prof. Hugo Miranda and Xavier Vilaça.

1.5 Structure of the Document

The rest of this document is organized as follows. For self-containment, Section 2 provides an introduction to all background related to our work. Chapter 3 describes the proposed architecture and its implementation. Chapter 4 presents the results of the experimental evaluation study. Finally, Chapter 5 concludes this document by summarizing its main points and future work.

2 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 2.1 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 2.2. Subsequently, in Section 2.3, 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(Agrawal, El Abbadi, and Steinke 1997). 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).

2.1 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(Feldman, Papadimitriou, Chuang, and Stoica 2006). 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 nodes, 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.

2.1.1 Payments

One possible mechanism to foster cooperation is to implement some payment scheme (Golle, Leyton-Brown, Mironov, and Lillibridge 2001; Yang and Garcia-Molina 2003). 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 secure central entity that all peers would trust.

2.1.2 Reciprocity

In a reciprocity-based system, a peer monitors other peers' behavior and evaluates their contribution based on their past actions (Menasché, Massoulié, and Towsley). There are two types of reciprocity-based schemes: direct and indirect.

2.1.2.1 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 (Cohen 2003) 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.

2.1.2.2 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.

2.1.3 Reputation

Reputation (Gupta, Judge, and Ammar 2003), 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 (Karakaya, Korpeoglu, and Ulusoy 2009). Reputation schemes can be local or global.

2.1.3.1 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.

2.1.3.2 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.

2.1.4 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.

2.1.4.1 Sybil Attack

A common exploitation of this a system with a very simple joining mechanism is the Sybil attack (Douceur 2002), where a user is able to forge multiple identites and uses them to gain a disproportionately large influence over the system. Systems with zero cost identities are the most affected by this attack as a user does not have any kind of validation or punishment. In order to prevent this attack, validation techniques can be used to dismiss masquerading hostiles entities. It is possible to use a certral entity which ensures and forces a one-to-one correspondence between an identity and an entity. This solution as the main drawbacks as the

ones describe in the Global Reputation approach. As (Douceur 2002) mentions, without any centralized authority, Sybil attacks are always possible except under unrealistic assumptions and unlimited resources. Even though, full prevention is very hard, it is possible to minimize the window of opportunity that a user had to exploit this attack, such as crypto-puzzles which its resolution time is predefined and that the node has to dedicate all computacional resources to its resolution (Wang and Reiter 2003; Merkle 1978; Borisov 2006).

2.1.4.2 Crypto-puzzles

are a good mechanims for mitigating the effects of undesirable network communication, such as flooding attacks. Crypto-puzzles can reduce the rhythm of creating new identities on the system and, therefore, prevent Sybil attacks. Crypto-puzzles should be hard to solve but easy to validate. This means that a client should spend a considerable amount of time and resources trying to solve it, while another client with a simple and fast operation should be able to validate if the cryto-puzzle was successfully solved.

2.1.4.2.1 Hash-Reversal Puzzles (Juels and Brainard 1999) approach is to force clients to reverse cryptographic hashes given the original random input with n bits hidden. The goal is to use a brute force search mechanism to discover the hidden bits. One advantage of this approach is how fast puzzles can be generated as single hash is performed very quickly. From the other hand, one main disadvantage of this approach is the fact that solution time is probabilistic. This means that a lucky client can solve the puzzle faster than a client who struggles at brute forcing the solution.

2.1.4.2.2 Hint-Based Hash-Reversal Puzzles (Feng, Kaiser, and Luu 2005) provides a single hash-reversal puzzle and a hint that gives the client an idea of where the asnwer of the puzzle is. The main purpose of the hint is to solve the main disadvantage of hash-reversal puzzles, where resolution time is not constant.

2.2 Membership Protocols

2.2.1 SCAMP

(Ganesh, Kermarrec, and Massouli 2001) 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.

2.2.2 HyParView

(Leitão, Pereira, and Rodrigues 2007) 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(Leitão, Pereira, and Rodrigues 2007) 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 sub-set 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.

2.3 Dissemination Protocols

2.3.1 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.

2.3.1.1 Basic Gossip

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 spread 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. However, gossip system 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, allowing a member to select, at random, a subset of nodes who 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 (Kermarrec, Massoulié, and Ganesh 2003) 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. 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.

2.3.1.2 Trees

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.

2.3.2 BitTorrent

BitTorrent (Cohen 2003), 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 who shared resources 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.

2.3.3 Dissemination with rational nodes

In reality, not all nodes are altruistic (Adar and Huberman 2000b). 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 (Li, Clement, Wong, Napper, Roy, Alvisi, and Dahlin 2006; Guerraoui, Huguenin, Kermarrec, Monod, and Prusty 2010), Trees (Levin, Sherwood, and Bhattacharjee 2006; Li, Xie, Qu, Keung, Lin, Liu, and Zhang 2008) and what strategies BitTorrent has to achieve the same goal.

2.3.4 BAR Gossip

BAR Gossip (Li, Clement, Wong, Napper, Roy, Alvisi, and Dahlin 2006) 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.

2.3.4.1 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.

2.3.4.2 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.

2.3.5 FlightPath

(Li, Clement, Marchetti, Kapritsos, Robison, Alvisi, and Dahlin 2008) 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.

2.3.6 LiFTinG

(Guerraoui, Huguenin, Kermarrec, Monod, and Prusty 2010) 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 sends 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.

2.3.6.1 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.

2.3.6.2 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 requests 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.

2.3.7 FOX

(Levin, Sherwood, and Bhattacharjee 2006) 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.

2.3.8 Coolstreaming

(Li, Xie, Qu, Keung, Lin, Liu, and Zhang 2008) 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.

2.3.9 BitTorrent

BitTorrent has two types of up-loaders: seeders and leechers (Locher, Moor, Schmid, and Wattenhofer 2006).

2.3.9.1 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.

2.3.9.2 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.

Summary

Peer-to-peer architecture are a great mechanism to spread the working load among users in the system. However, due the hard task of controling and ensuring that every member of the system is acting faithfully to the specified protocol, some nodes might avoid contributing to the system and still benefit from it. In this chapter we discussed the most popular systems that ensure that nodes follow the protocol or penalize them if a different behavior is adopted.

Furthermore, we discussed two main approaches of supporting live-streaming systems and prototypes already developed that use heavy control mechanisms or cipher overhead to guarantee the correct and expected results.

The next chapter will introduce the architecture and implementation details of our system FastRank.

3 FastRank

In this work 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(Huang, Fu, Chiu, Lui, and Huang 2008) among others(Liao, Jin, Liu, Ni, and Deng 2006). In this context, existing work (Li, Clement, Wong, Napper, Roy, Alvisi, and Dahlin 2006; Li, Clement, Marchetti, Kapritsos, Robison, Alvisi, and Dahlin 2008; Guerraoui, Huguenin, Kermarrec, Monod, and Prusty 2010) 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

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.

3.1 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) (Kermarrec, Massoulié, and Ganesh 2003), 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(Leitão, Pereira, and Rodrigues 2007) 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 (Leitão, Pereira, and Rodrigues 2007). 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.

3.1.1 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(Wang and Reiter 2003; Merkle 1978; Borisov 2006); in FastRank we have opted to use (Feng, Kaiser, and Luu 2005).

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).

3.2 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(Karakaya, Korpeoglu, and Ulusoy 2009). 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 4, we discuss how an appropriate value for *minrank* can be selected.

3.3 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 4 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 $fp_i(j)$:

$$fp_i(j) = \begin{cases} bfp & \text{if } rank_{ij} \geq baserank \\ bfp \left| \frac{minrank - rank_{ij}}{minrank} \right| & \text{if } baserank > rank_{ij} \geq minrank \end{cases}$$

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*.

FastRank was designed for systems where most of the nodes are altruistic and just a small fraction are free riders or racional. As described in the related work, the scenario where all nodes are rational required heavy and high overhead protocols, therefore it is out of the scope of FastRank. On evaluation section, we will analyze the streaming quality for different fraction of nodes that follow different protocols, considering that nodes belong to one of the following categories:

- *Altruistics:* An altruistic node will always follow the specified protocol and never adopts a different one.

- *Free Rider:* Are nodes that do not forward any message. They receive information from their neighbours until the moment they are expelled from their views. Besides, they try to maximize the amount of neighbours that they are connected to.
- *Rational:* Are nodes that try to deviate from the protocol in order to maximize their utility. This behavior is discussed below.

3.4 Rational Behavior

We consider that a rational node is benefited everytime it receives a new packet and has a cost everytime it forwads it or solves a crypto-puzzle. Therefore, in order to maximize its utility, a racional tries to increase the ratio of received and sent packages. It is clear from the description of the algorithm that, if a node does not maintain a balanced ration of received and sent packages, it will, eventually, be considered as a free rider and expelled by that neighbor. There are two possible strategies that a rational node might have to increase its utility:

- *Decreasing the size of the active view:* A rational node can opt for decreasing the size of its active view, in order to keep receiving the information and forwarding the information to a inferior amount of neighbors. With this behavior, the node appears to be altruistic for its neighbors, that do not have a way of detecting that the node is using a reduced view size. Sophisticated algorithms (e.g. LiFTinG (Guerraoui, Huguenin, Kermarrec, Monod, and Prusty 2010)) that do not only rely on local information, were designed to detect this attack.
- *Minimum service (keep the score strictly above the threshold):* A rational node can reduze its forwarding message rate in order to maintain its own score just above the minimal threshold and, therefore, never be expelled from the active view of altruistic nodes. Beseides, it might try to find as many neighbors as possible. The main goal is to try to connect to enough neighbors, such that it can receive good quality information, although rarely forward any information.
- *Sybil attack: always accept new connections* As described before, Sybil attacks are almost impossible to prevent without the use of a central authority. Even though in our system we try to prevent this attack by adding a cost and limiting the node capacity to establish new

connections, a node might create a sybil identity and forward every join request to that node which accepts unlimited connections. The idea is to avoid the cost of establishing a new connection, while being able to have as much connections as possible.

We will show that, given the above deviations, FastRank has the following properties: i) the reliability of an altruisitc node is only partly affected, even with a considerable fraction of nodes with non altruistic behavior; ii) rational nodes still have to contribute with a reasonable amount of resources in order to be able to receive the stream in an acceptable quality; iii) if there is a big fraction of nodes adopting a rational behavior, it can be detected by the altruistics nodes.

Summary

In this chapter we described the design and implementation of FastRank. Since supporting an arbitrary network topology is not only hard, but requires heavy mechanisms, we create a topology where nodes maintain symmetric connections. This property eliminates the need of having a central authority or a set of peers that are responsible for controling and validating the behavior of every member in the system. Instead, nodes are able to directly check the behavior of their neighbors in a simple and effective way. Using tit-for-tat strategies, nodes tend to benefit the neighbors who have provided them information on the past, while free-riders will keep a low probability of receiving any further update. Crypto-puzzles are used in order to limit the speed that a node is able to establish new connections and incentive nodes to preserve their actual connections, which also works as a Sybil attack prevention.

In the next chapter we present the experimental evaluation made using this prototype.

4 Evaluation

In this section, we provide an extensive evaluation of FastRank. All experiments were performed using the PeerSim Framework (Montresor and Jelasity). 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.

4.1 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 4.1 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).

4.1.1 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

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

Table 4.1: Default configuration of FastRank

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 4.1 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.

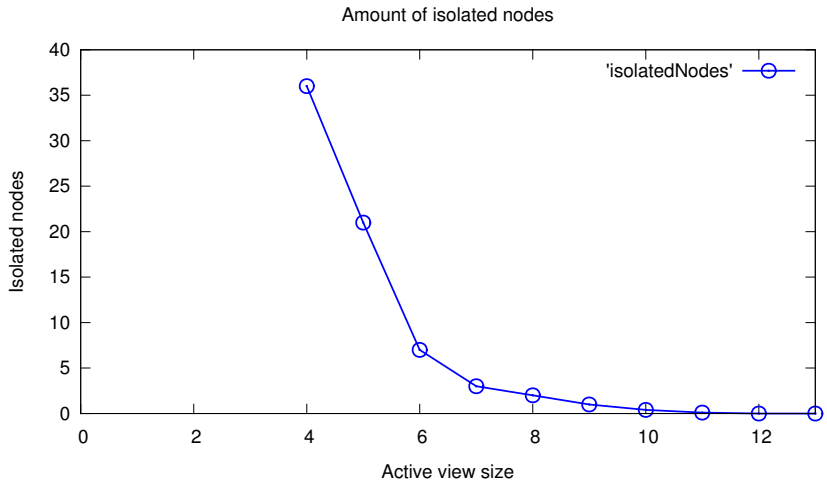
4.1.2 Forward Probability

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 4.2 depicts the reliability of the streaming protocol on a FastRank overlay, as a function of the dissemination probability and Figure 4.3 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.

4.1.3 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

Figure 4.1: Isolated nodes after 30% failures.

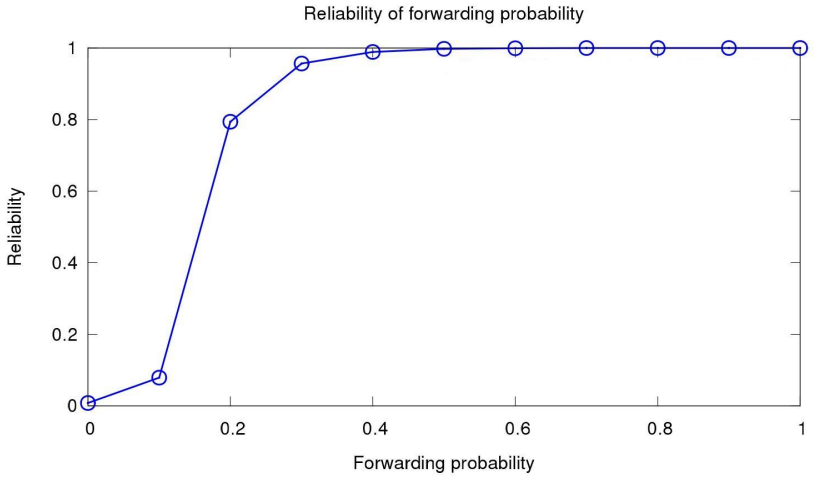


that the rank of altruistic nodes also drops below *minrank*, a situation that we call a *false positive*. Our goal is then to promptly detect free-riders while minimising false positives. Therefore, the value of *minrank* should weigh this trade-off. Figure 4.5 shows the fluctuation of the rank among 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 4.6. 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.

4.1.4 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 4.6 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.

Figure 4.2: Reliability.

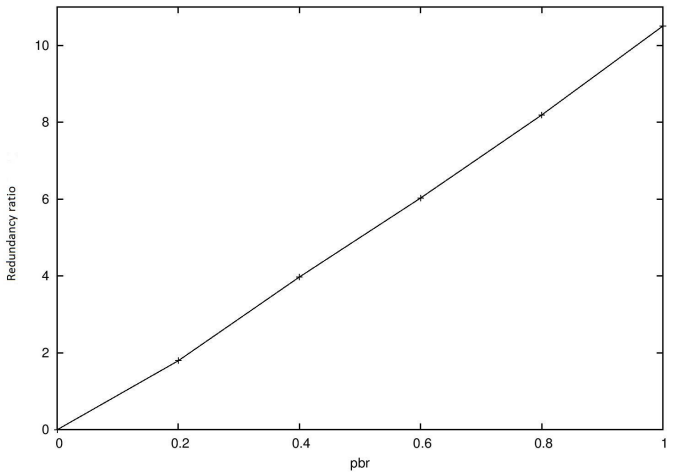


4.2 Failure-free Operation

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 4.2.

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

Figure 4.3: Redundancy.



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

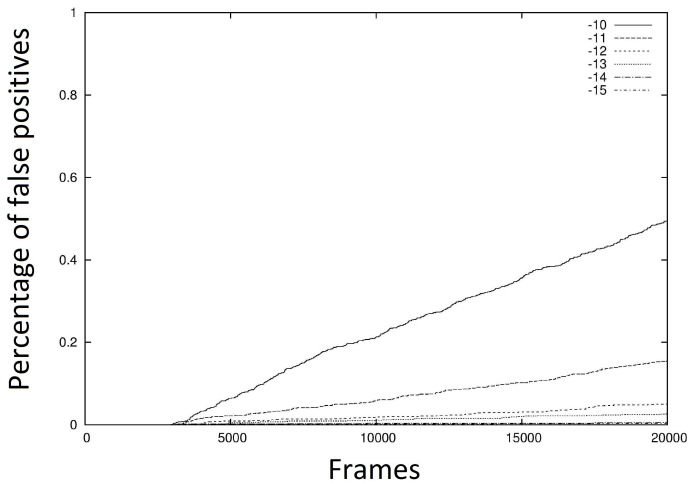
Table 4.2: Operation in Absence of Misbehaviour

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.

4.3 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 4.7 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

Figure 4.4: False positives.



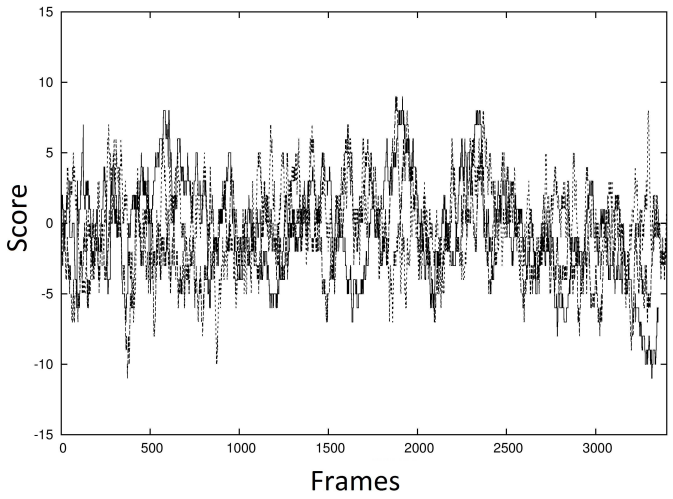
view of an altruistic node are other altruistic nodes. Consequently, free-riders have become disconnected from the network.

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.

4.4 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

Figure 4.5: Rank fluctuation.



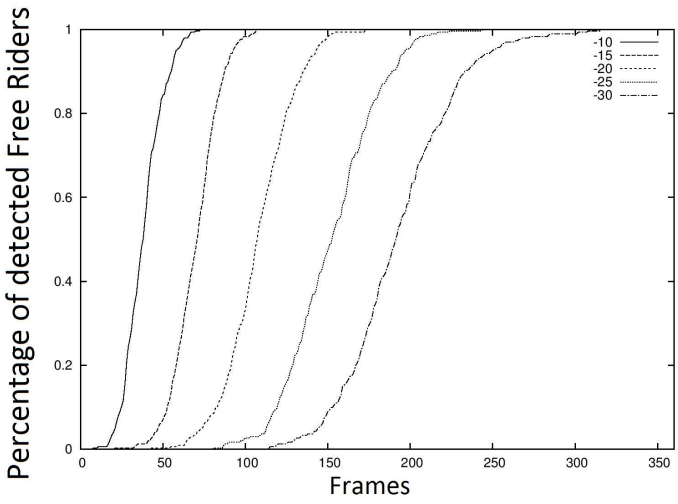
is the difference between the benefits and the costs of executing the protocol. We consider that the only significant costs of our protocol are those of forwarding frames and computing crypto-puzzles. We also assume that the fraction of rational nodes is constant.

4.4.1 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-

Figure 4.6: Detection Time.



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 altruistic nodes.

4.4.2 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

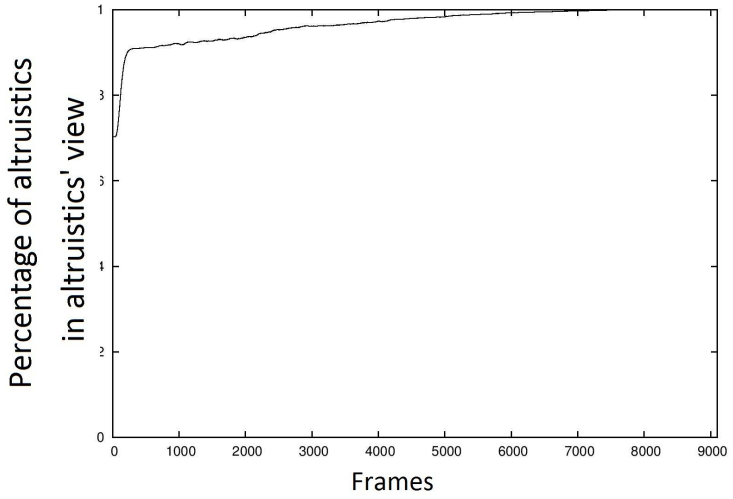


Figure 4.7: Recovering from free-riders

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 4.8 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.

4.4.3 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 4.3, 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

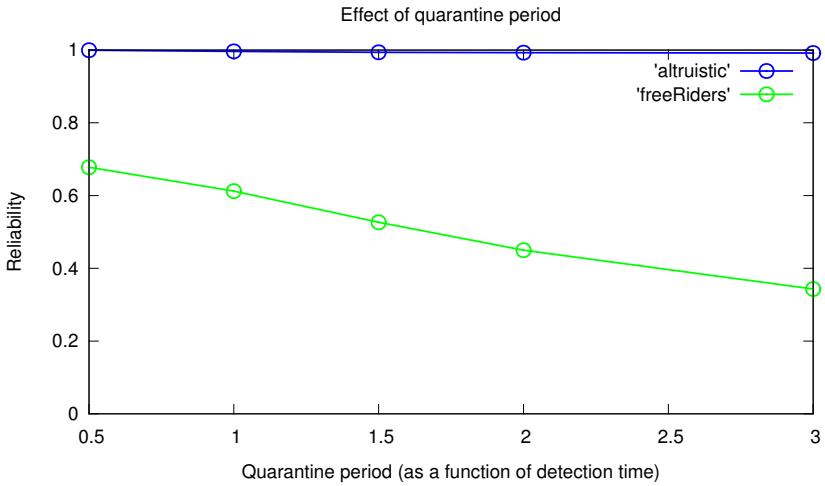


Figure 4.8: Effect of quarantine period

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 4.3: Effect of altruistic with shrunk view strategy (30% of rational nodes)

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 (Guerraoui, Huguenin, Kermarrec, Monod, and Prusty 2010)).

4.4.4 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 4.4 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.

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 4.4: Effect of minimal forwarding with same view strategy (30% of rational nodes)

4.4.5 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 4.4 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.

4.4.6 Forward join request to a Sybil identity

In order to minimize the amount of solved crypto-puzzles, a node might create a sybil identity and forward every join request that it receives to that node who accepts any connection. Table 4.5 shows that this attack only is successful if done at the very beginning of the topology organization. This happens not only because nodes tend to search for new connections at the beginning rather then after the topology is stable, but also because at the beginning nodes tend to interact with an higher percentage of total nodes of the system, which increases the probability of requesting a join through the attacker.

4.5 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

	Amount of neighbors
Beginning of topology organization	105
Middle of the topology organization	42
After topology organization	15

Table 4.5: Effect of forwarding the join requests to a Sybil identity

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 4.6: Effect of minimal forwarding with enlarged view strategy (for one rational node)

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.

Summary

In this chapter we introduced the experimental evaluation made to FastRank and its results. We started by motivating the choice of the selected values for the different parameters of the system. Next, we showed how does the system operates when all nodes are altruistic. It showed that FastRank is able to provide a reliability of 0.999% to all members of the system, including newcomers. In the situation where not all nodes are altruistic, we showed that the system is able to recover from free riders and an altruistic node is able to keep receiving the information with good quality and will, eventualy, detect the free riders that it is connected to and search for new connections in order to find a new altruistic neighbor. On the last part of the evaluation, we showed the impact of rational nodes in the system. Even though our system is vulnerable to certain types of attacks, we showed that its benefit its limited and incurs extra computacional

cost, sophisticated strategies or a perfect timing in order to achieve success.

The next chapter finishes this thesis by presenting the conclusions regarding the work developed and also introduces some directions in terms of future work.

5 Conclusions

5.1 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.

5.2 Future Work

As future work we would like to able to support three phase gossip mechanims while maintaing the properties we are able to provide, i.e. excellent reliability, minimal false-positives and fast free-rider behavior detection. This would reduce the overall message size distributed among the system, but more sophisticated rational attacks would arise. We would also like to study the impact of a dynamic population. Nodes in a dynamic population might arbitrary swap between behaviors and this change might occurs in different points in time for different nodes. This would simulate in a more realistic way what happens in practical situations of peer-to-peer

networks. Considering that FastRank might not provide the best reliability for the users if a big percentage of node are rational or free riders, we would like to, based on a set of predefined metrics and rules, notify a central authority that would require nodes to run a different protocol in order to ensure and maintain a good service quality, i.e. the protocol would adapt based on how willing the population is to share their resources.

References

Adar, E. and B. Huberman (2000a, October). Free riding on gnutella. *First Monday 5*(10).

Adar, E. and B. A. Huberman (2000b). Free riding on gnutella.

Agrawal, D., A. El Abbadi, and R. C. Steinke (1997). Epidemic algorithms in replicated databases. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pp. 161–172. ACM.

Bonald, T., L. Massoulié, F. Mathieu, D. Perino, and A. Twigg (2008). Epidemic live streaming: optimal performance trade-offs. In *ACM SIGMETRICS Performance Evaluation Review*, Volume 36, pp. 325–336. ACM.

Borisov, N. (2006). Computational puzzles as sybil defenses. In *IEEE P2P 2006*, pp. 171–176.

Cohen, B. (2003). Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, pp. 68–72.

Douceur, J. (2002). The sybil attack. In *Peer-to-peer Systems*, pp. 251–260. Springer.

Feldman, M., C. Papadimitriou, J. Chuang, and I. Stoica (2006). Free-riding and whitewashing in peer-to-peer systems. *Selected Areas in Communications, IEEE Journal on 24*(5), 1010–1019.

Feng, W.-C., E. Kaiser, and A. Luu (2005). Design and implementation of network puzzles. In *IEEE INFOCOM 2005*, Volume 4, pp. 2372–2382.

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

Golle, P., K. Leyton-Brown, I. Mironov, and M. Lillibridge (2001). Incentives for sharing in peer-to-peer networks. In *Electronic Commerce*, pp. 75–87. Springer.

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

Gupta, M., P. Judge, and M. Ammar (2003). 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*, pp. 144–152. ACM.

Huang, Y., T. Fu, D.-M. Chiu, J. Lui, and C. Huang (2008). Challenges, design and analysis of a large-scale p2p-vod system. *SIGCOMM Comput. Commun. Rev. 38*(4), 375–388.

Hughes, D., G. Coulson, and J. Walkerdine (2005, June). Free riding on gnutella revisited: The bell tolls. *IEEE Distributed Systems Online 6*(6), 1–.

Jenkins, K., K. Hopkinson, and K. Birman (2001). A gossip protocol for subgroup multicast. In *Distributed Computing Systems Workshop, 2001 International Conference on*, pp. 25–30. IEEE.

Juels, A. and J. G. Brainard (1999). Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *NDSS*, Volume 99, pp. 151–165.

Karakaya, M., I. Korpeoglu, and O. Ulusoy (2009). Free riding in peer-to-peer networks. *Internet Computing, IEEE 13*(2), 92–98.

Kermarrec, A.-M., L. Massoulié, and A. J. Ganesh (2003). Probabilistic reliable dissemination in large-scale systems. *IEEE Trans. on Parallel and Distributed Systems, 14*(3), 248–258.

Leitão, J., J. Pereira, and L. Rodrigues (2007, October). Epidemic broadcast trees. In *IEEE SRDS*, Beijing, China, pp. 301–310.

Leitão, J., J. Pereira, and L. Rodrigues (2007). Hyparview: A membership protocol for reliable gossip-based broadcast. In *IEEE DSN*, pp. 419–428.

Levin, D., R. Sherwood, and B. Bhattacharjee (2006). Fair file swarming with fox. In *IPTPS*.

Li, B., S. Xie, Y. Qu, G. Y. Keung, C. Lin, J. Liu, and X. Zhang (2008). Inside the new coolstreaming: Principles, measurements and performance implications. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*. IEEE.

Li, H., A. Clement, M. Marchetti, M. Kapritsos, L. Robison, L. Alvisi, and M. Dahlin (2008). Flightpath: Obedience vs. choice in cooperative services. In *OSDI*, Volume 8, pp. 355–368.

Li, H., A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin (2006). Bar gossip. In *OSDI*, pp. 191–204.

Liao, X., H. Jin, Y. Liu, L. Ni, and D. Deng (2006). Anysee: Peer-to-peer live streaming. In *INFOCOM*, Volume 25, pp. 1–10.

Locher, T., P. Moor, S. Schmid, and R. Wattenhofer (2006). Free riding in bittorrent is cheap. In *Proc. Workshop on Hot Topics in Networks (HotNets)*, pp. 85–90. Citeseer.

Menasché, D. S., L. Massoulié, and D. Towsley. Reciprocity in peer-to-peer systems.

Merkle, R. (1978). Secure communications over insecure channels. *Comm. of the ACM 21*(4), 294–299.

Montresor, A. and M. Jelasity. Peersim: A scalable p2p simulator *.

Stutzbach, D. and R. Rejaie (2006). Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pp. 189–202. ACM.

Wang, X. and M. Reiter (2003). Defending against denial-of-service attacks with puzzle auctions. In *Security and Privacy*, pp. 78–92. IEEE.

Yang, B. and H. Garcia-Molina (2003). Ppay: micropayments for peer-to-peer systems. In *Proceedings of the 10th ACM conference on Computer and communications security*, pp. 300–310. ACM.