EquiCast: Scalable Multicast with Selfish Users*

Idit Keidar
EE Department, Technion
idish@ee.technion.ac.il

Roie Melamed
IBM Haifa Research Labs
roiem@il.ibm.com

Ariel Orda
EE Department, Technion
ariel@ee.technion.ac.il

Abstract

Peer-to-peer (P2P) networks suffer from the problem of "freeloaders", i.e., users who consume resources without contributing anything in return. In this paper, we tackle this problem taking a game theoretic perspective by modeling the system as a non-cooperative game. We introduce EquiCast, a wide-area P2P multicast protocol for large groups of selfish nodes. EquiCast is the first P2P multicast protocol that is *formally proven* to enforce cooperation in *selfish environments*. Additionally, we prove that EquiCast incurs a low constant load on each user.

Categories and Subject Descriptors: C.2.4 [Computer-Communication Networks] Distributed Systems—*Distributed applications*.

General Terms: Algorithms, Theory.

Keywords: Peer-to-peer networks, multicast, non-cooperative game, incentives.

Contact Author: Roie Melamed, `roiem@il.ibm.com`, IBM Haifa Research Labs,
University Campus, Haifa, 31905, Israel, Tel: +972-4-828-1023.

*A preliminary version of this paper appears in Proceedings of the 25th ACM Symposium on Principles of Distributed Computing (PODC), July 2006, Denver, Colorado.

1 Introduction

Peer-to-peer (P2P) networks can distribute digital content to a large number of users over the Internet by distributing the load among the peers [6, 24]. However, these networks suffer from the problem of "freeloaders", i.e., users who consume resources without contributing their fair share [3]. In order to discourage freeloaders, some P2P systems employ incentives to motivate users to cooperate, e.g., contribute upload bandwidth or disk space for some other users. However, while current incentive-based P2P systems reward cooperation to some extent, no existing protocol has been proven to enforce cooperation in selfish environments. Moreover, such systems, e.g., [6, 11], typically rely to some extent on the pure altruism of "seeder" nodes, which are expected to upload data blocks to other nodes for no return whenever they have available bandwidth [6, 11, 13, 20]. Hence, current incentive-based P2P systems do not solve the problem of freeloaders [13, 26], and would not have worked well at all if users would have behaved selfishly, e.g, leaving a content distribution system after they have finished downloading the file [11, 13].

Nowadays, user altruism is common since most users are connected to the Internet using static machines via ISPs with a flat pricing model, and hence sending a packet does not incur a cost on its sender. However, these paradigms are changing. First, the increasing access to digital content is expected to drive ISPs to implement a tiered pricing scheme, where high end pricing plans will allow unlimited downloads and uploads, while lower tier pricing plans will limit traffic bandwidth [24]. With such a pricing scheme, users will most likely cease to be altruistic [24]. This may lead to low P2P system availability [13, 16] or even system collapse [3, 23]. Second, wireless hotspots are proliferating in recent years, and users are increasingly connecting to the Internet and downloading content to mobile devices such as laptops and cell phones. In such networks, pricing is typically based on connection time or transmission volume. Moreover, battery power is a critical resource for mobile devices. Hence, user altruism can hardly be expected in such networks. Therefore, we believe that it is important to design P2P systems that work well even when all users are selfish.

In this paper, we address this challenge. We introduce *EquiCast*, a wide-area P2P multicast protocol for distributing content to large groups of selfish nodes. We treat the problem of freeloading from a game theoretic perspective, and we model the system as a *non-cooperative game*. In such a game, nodes are selfish but *rational*, i.e., each user chooses its own *strategy* regarding its level of cooperation so as to minimize its own cost [10]. More specifically, the goal of each node is to receive all the multicast packets while minimizing its sending rate. We define a special set of *protocol-obedient strategies (POSs)*. Generally speaking, a strategy out of this set allows a node to determine how many connections to maintain and how

many packets to send on each connection though it forbids hacking the protocol's code or assuming that other nodes hack the protocol's code. We prove that if all nodes choose POSs, then each node receives all the multicast packets and, moreover, no node can unilaterally reduce its cost by changing its strategy to a non-POS. That is, unilateral hacking of the protocol's code cannot reduce a node's cost. In addition, we believe that it is reasonable to assume that most nodes will run a POS, since users usually either do not have the required technical knowledge to modify an application code or they do not download a hacked code as there is a risk in downloading such a code. Our formal model and cost function are presented in Section 3, and in Section 5.2 we formally define the set of POSs.

In EquiCast, a single distribution server *S* (which can be implemented by multiple machines acting as one logical server) organizes the nodes into an *overlay network*. We divide the time into rounds, and in each round, *S* injects new data packets to a small random subset of the nodes in the overlay. Nodes, then, communicate with their overlay neighbors in order to retrieve missing data packets. *S* also provides a "safety net" for a node whose data receiving rate is lower than the multicast rate, by sending data packets to either the node or its neighbors. This additional overhead incurred on *S* is modest, since most of the nodes are expected to receive most if not all the multicast packets from their overlay neighbors.

EquiCast enforces cooperation through two mechanisms. The first is a *monitoring mechanism*, whereby each node monitors the sending rate of each of its neighbors. Specifically, for each neighbor $\hat{n}$, each node n maintains $\hat{n}$'s *balance*, which is the difference between the number of data packets $\hat{n}$ has sent to n so far and the expected per-link throughput. As long as $\hat{n}$'s balance is greater than or equal to a predefined negative threshold L, $\hat{n}$ is considered to be cooperative, and n continues to send data packets to $\hat{n}$. Otherwise, n terminates its connection with $\hat{n}$. Note, however, that it is always possible for cooperative nodes to have a balance greater than or equal to L with respect to all of their neighbors.

The second mechanism is a per-link *penalty mechanism*, which further motivates nodes to adhere to the expected link throughput. It charges a neighbor with one additional *fine* packet for every round the neighbor has a negative balance, where a fine packet is a dummy packet that has the same size as a data packet. Fine packets, in contrast to data packets, do not affect the node's balance. Therefore, a node is motivated to achieve a non-negative balance, whenever possible. Note that the multicast rate is tens of data packets per round, and hence the penalty mechanism incurs a modest overhead. In general, in EquiCast, each node is required to have an upload bandwidth that is slightly higher, e.g., by 10%, than the multicast rate. Note that similar requirements are also assumed by multicast systems for cooperative environments [4].

In Section 5, we prove that, in environments in which all the nodes are selfish, if all the nodes choose

POSs, then EquiCast disseminates all the multicast packets to all the nodes. Additionally, every POS in which a node exclusively cooperates with all its neighbors *strictly dominates* every POS in which it does not. This means that the cost incurred on a freeloader node is strictly higher than the cost incurred on a cooperative node, and hence all nodes are expected to follow the protocol out of their own selfish interests. Finally, we prove that if all the nodes choose POSs, then no node can unilaterally reduce its cost by changing its strategy to a non-POS. Moreover, we show that this result holds under some milder conditions, where some of the nodes might adopt non-POSs or even be non-rational. We are unaware of any previous P2P multicast protocol that was formally proven to enforce cooperation in environments in which *all* nodes are selfish.

Finally, for simplicity, throughout most of the paper we describe only a static version of EquiCast, in which no node joins or leaves the service. In Section 6, we sketch out a dynamic version of EquiCast that supports node joins and leaves.

2 Related Work

EquiCast's proof of cooperation enforcement relies on i) the centralized construction of the overlay by a single distribution server and ii) the assumption that a node cannot change the protocol's code. Besides EquiCast, we are familiar with only two previous P2P multicast protocols for selfish environments [12, 23].

Similarly to EquiCast, Ngan et al. [23] propose an incentive-based multicast protocol that relies on the above two assumptions. The protocol in [23] is based on detection of selfish nodes and periodic reconstruction of multicast trees that exclude previously misbehaving nodes. However, this protocol induces high overhead. For example, with 500 nodes, the trees' reconstruction requires each node to send 256 control messages every two minutes; and when the group size is 2000 nodes, each node sends nearly 400 control messages every two minutes, in addition to data messages. As opposed to EquicCast and the protocol in [23], Habib and Chuang [12] propose an incentive-based protocol for media streaming that does not rely on the above two assumptions. In [12], cooperative nodes receive high quality of service whereas freeloaders receive low quality streaming. While this protocol rewards cooperation to some extent, it does not solve the problem of freeloaders. These two solutions, however, consider a different model, in which only a fraction of the nodes are selfish. Moreover, neither is formally proven to enforce cooperation.

The only distributed Internet service we are aware of that is proven to enforce cooperation in selfish environments is the FOX file sharing service [20], which was developed in parallel to our work. FOX employs an effective and robust variant of the tit-for-tat strategy, and provides theoretically optimal download times

when all the nodes cooperate. However, in FOX, as opposed to EquiCast, the cost of achieving fairness is high: a violation of the protocol by a single node causes to the collapse of the whole system [20, 21]. In addition, the study in [20] does not model the system as a non-cooperative game as we do.

Other previous distributed Internet services such as content distribution [6, 11], storage [7], and lookup [5] reward cooperation to some extent by incentivizing cooperative behavior. The BitTorrent [6] and Avalanche [11] content distribution systems support the tit-for-tat strategy, in which a user preferentially uploads blocks of information to users from which it is also downloading blocks. But these systems rely to some extent on user altruism, and hence they do not purport to work in a selfish environment where all users are rational and selfish, and every packet incurs a cost on its sender. In the SAMSARA storage system [7], each node is required to contribute as much disk space to the system as it is using, and in the GIA lookup system [5] the quality of service experienced by a node is proportional to its contribution to the system. None of the aforementioned services, however, models the system as a non-cooperative game or formally proves cooperation as we do.

In P2P protocols based on a centralized *reputation system*, e.g., eMule [1], and [3], each node sends to and requests from the system reports about the level of cooperation of other nodes. Hence, a node is motivated to collaborate with other nodes. However, this approach achieves limited scalability [3], since the reputation system continuously communicates with all the nodes.

The BAR-B backup service [2] can tolerate both Byzantine nodes and an unbounded number of selfish nodes by using asynchronous replicated state machines. The replicated state machine approach, however, incurs high overhead [21], and hence can support only a limited number of nodes. Similarly to EquiCast, in the BAR Gossip protocol [21] a node gossips with other nodes in order to retrieve missing data packets. The BAR Gossip protocol guarantees predictable throughput and low latency either in a setting in which up to 40% of nodes are selfish and collude and the rest follow the protocol or in a setting in which a node is either selfish or Byzantine and up to 20% of the nodes are Byzantine. In contrast to EquiCast in which a node gossips with a fixed small set of nodes (the node's overlay neighbors) received from the centralized server, in BAR Gossip the centralized server sends to each node the full static membership list, and each node chooses gossip partners out of this list. In addition, in BAR Gossip, prior to a broadcast session the centralized server is required to create a public/private key to each node and to send the full list of public keys to all the nodes. Disseminating the full membership and cryptographic information to all the nodes induces quadratic message complexity on the centralized server, which limits the scalability. Finally, the BAR Gossip protocol assumes a static setting with no membership changes, whereas in Section 6 we present a dynamic version

of EquiCast, which can be deployed in a dynamic setting.

The Dandelion content distribution system [27] provides robust incentives for clients who possess content to serve others. Specifically, when a node n_1 uploads to a node n_2, n_1 sends encrypted content to node n_2. To decrypt, n_2 must request keys from the centralized server. The requests for keys serve as the "proof" that n_1 has uploaded some content to n_2. Thus, when the server receives a key request, it credits n_1 with credit that can be redeemed for future downloads, and charges n_2 for the content. Such a scheme motivates cooperative behavior, though it incurs high overhead on the centralized server, since each data exchange between two nodes results in communication with the centralized server. Additionally, as opposed to EquiCast, Dandelion was only evaluated experimentally and was not proven to enforce cooperation in a setting in which all the nodes are selfish.

SecureStream [14] is an application-level live streaming system built upon a Byzantine membership protocol and pull-based packet dissemination architecture. In SecureStream a node can download packets from multiple other nodes, and hence nodes are not dependent on any particular node, which increases the fault-tolerance to malicious nodes. Experimentally, SecureStream was shown to tolerate a limited percentage of malicious nodes, and it achieves graceful degradation in the presence of increasing ratios of attackers. In contrast, in this paper we focus on formal proofs, and we formally prove that EquiCast enforces cooperation in selfish environments.

Herman and Johnen [15] studied P2P self-downloading protocols, in which a node disconnects upon file download completion, though it cooperates (i.e., allow other nodes to download data packets from it) as long as the file download is not completed. Herman and Johnen proved that self-downloading is feasible if there is a stream of nodes that arrive, share while downloading, and exit when finished, provided the arrival rate is sufficient to replace departing nodes, and the download bandwidth is the is the same for all nodes. In contrast, we assume that all the nodes are selfish throughout the entire download session, and we prove that in such a setting EquiCast enforces cooperation.

Finally, *cost-sharing* multicast solutions e.g., [8], consider a different model, in which multicast is provided over a dedicated infrastructure, and the infrastructure cost is shared among all nodes. Such an approach, however, is not applicable to P2P systems.

3 Model and Problem Statement

We consider a large static collection of N nodes n_1, n_2, ..., n_N. A single distribution server S distributes P data packets to the nodes, where P is a random variable distributed exponentially with a large expectation,

e.g., larger than $10{,}000$. The value of P is not known to the nodes. *S* knows all the node identities, e.g., by each node registering itself at *S*.

As many previous studies, e.g., [4, 18], we assume a single distribution server. However, EquiCast can support a small number of additional distribution servers by having each such additional distribution server send the packets it disseminates to the distribution server *S*.

3.1 Network and timing model

Each node can directly communicate with every other node and with *S*. The multicast rate is p data packets per δ time units. Each node has an upload bandwidth of $p{+}kc$ packets per δ time units, where k and c are small constants such that $k{\geq}3$, $c{\geq}4$, and $p\%k{=}0$ (a node that runs a POS has k neigbhors, and to each such neighbor it sends no more than $\frac{p}{k} + c - 3$ packets every δ time units). In addition, we require that $(k^2{-}k)(c{-}3){<}p$ and $k^2(c{-}2){-}2k{<}p$, in order to prove that every POS in which a node exclusively cooperates with all its neighbors strictly dominates every POS in which it does not. There is a bound of Δ time units on packet delay, and sending a packet incurs zero delay on the sender. Local computations also incur zero delay. Finally, for simplicity, we assume no packet loss.

3.2 The game formulation

We model the system as a *non-cooperative game*, in which the *players* are the N nodes. Each node chooses a *strategy* that dictates how it plays the game. A *strategy profile* is a vector of N strategies, one for each node. A *strategy space* is the set of strategies available to each node. In this paper, the strategy space consists of the set of all the strategies that send no more than $p{+}kc$ packets per δ time units.

We define a special set of *protocol-obedient strategies (POSs)*. Generally speaking, a strategy out of this set must run the protocol as is and can only determine how many connections to maintain and how many packets to send on each connection. In Section 5, we prove that if a node n cannot locate an identity of a node that does not choose a POS, i.e., n cannot maliciously collude with some other malicious nodes, then n follows the protocol in order to reduce its selfish cost. This is since EquiCast's monitoring mechanism enforces n to send data packets to a given neighbor (that chooses a POS) at the expected per-link throughput in order to get data packets from this neighbor. We defer the precise definition of this set to Section 5.2, since it relies on the protocol's code described in Section 4.3.

Each node is selfish and *rational*, i.e., n_i chooses a strategy st_i that minimizes its individual cost as defined below. A strategy st *strictly dominates* another strategy st' if choosing st always incurs a lower cost

than choosing st', regardless of the strategies chosen by other nodes. A *strongly dominant strategy* strictly dominates all other strategies. Although *S* is not one of the players, we model its random injections of data packets as its strategy st_0, and hence our proof of cooperation is valid regardless of *S*'s random choices. Note that st_0 does not determine the length of the session, i.e., P. Denote by r_i the total number of data packets received by n_i throughout the multicast session, and by s_i the total number of packets sent by n_i throughout the multicast session. Then, the cost function for a node n_i is defined as:

$$f_i(st_0, st_1, ..., st_N) = \begin{cases} \infty & \text{if } r_i < P \\ s_i & \text{if } r_i = P. \end{cases}$$

That is, if n_i receives all the multicast packets, then its cost is the number of packets it has sent during the multicast session. Otherwise, n_i's cost is infinite.

3.3 Problem statement

Our goal is to design a scalable P2P multicast protocol, in which if all nodes choose POSs, then (i) each node receives all the multicast packets; and (ii) no node can reduce its cost by unilaterally changing its strategy to a non-POS. A second goal is efficiency. The maximal total receiving and sending overhead incurred on each node throughput the entire multicast session is $P(1+\frac{ak}{p})$ and $P(1+\frac{ak}{p})+Hk$ packets, respectively, where a is a small constant and H is a non-negative constant determined by each node.

4 EquiCast

Section 4.1 describes EquiCast's architecture. Section 4.2 provides a high-level description of EquiCast's cooperation enforcement scheme, and Section 4.3 describes the protocol in detail.

4.1 Architecture

S organizes the nodes into a static overlay that satisfies the following properties: (KRRG1) each node in the overlay has exactly k neighbors for some parameter k; (KRRG2) the overlay's diameter is logarithmic in N; and (KRRG3) the expected distance between a given node and a random node in the overlay equals the average distance between a pair of nodes in the overlay. We note that only property KRRG1 is required in order to prove EquiCast's cooperation enforcement scheme, whereas properties KRRG2 and KRRG3 are expected to achieve low latency of event delivery and low load on *S*, respectively. For $k\geq 3$, a *k-regular*

random graph[1] always satisfies property KRRG1 and also satisfies properties KRRG2 and KRRG3 with high probability [9, 17, 29]. *S* constructs a k-regular random graph, e.g., using one of the constructions in [28, 29], and sends to each node the identities of its overlay neighbors, henceforth, simply called *neighbors*. Note that the expected construction time of a k-regular random graph is $O(Nk^2)$ [28], and hence such a centralized construction is feasible even in a large static setting. As described in Section 6, in a dynamic setting *S* can construct and maintain a k-regular overlay, e.g., [19], in which a leave/join operation incurs a processing time of $O(1)/O(log_{\frac{k}{2}}N)$. Hence, such a construction is feasible even in a large dynamic setting. Note also that since the construction is centralized, no node cooperation is required.

In the next section, we show that, under our model assumptions, for each node, maintaining connections with its k neighbors is a dominant strategy. Hence, connections are expected to persist. However, if a given connection is terminated, e.g., due to a node failure, then a node n can end up with less than k neighbors. In such cases, n contacts *S*, and *S* emulates a selfish rational EquiCast node $\hat{n}$, and a new connection is formed between n and $\hat{n}$. $\hat{n}$'s interface is identical to the interface of each EquiCast node with the following two exceptions: i) n's balance with respect to $\hat{n}$ is initialized to the lowest possible balance, i.e., L; and ii) in each round, n must send a fine packet to $\hat{n}$ regardless of its balance with respect to $\hat{n}$, otherwise $\hat{n}$ terminates its connection with n. Hence, as we show in Section 5.2, a node prefers to maintain a connection with a non-emulated node over an emulated one.

4.2 Overview

We divide the time into $R{=}\lceil\frac{P}{p}\rceil$ rounds. Every round, *S* creates p new data packets, and sends all the copies of these p packets to a random set of k nodes out of the N nodes.

In each round, every node n gossips with its neighbors about new data packets it has received in the previous round, i.e., for each neighbor, n sends a gossip packet containing the identities of all the data packets it has received in the previous round. After receiving gossip packets from its neighbors, n requests from each of its neighbors data packets that the neighbor has and were not previously received by n. If a given packet is available at more than one neighbor, then n randomly picks one of those neighbors to request the packet from. Finally, n sends its neighbors the data packets they requested from it.

We note that since a given packet is sent by *S* to each of the nodes with equal probability and since the expected distance between a random node and a given node equals the average distance between a pair of nodes in the overlay (KRRG3), if all the nodes comply with the protocol, then the average latency with

[1] A *k-regular random graph* with N nodes is a graph chosen uniformly at random from the set of k-regular graphs with N nodes.

which nodes receive data packets is identical for all nodes, and the expected throughput is $\frac{p}{k}$ data packets per-round on every overlay link. In a previous study [22], we used a similar technique in order to support reliable multicast in cooperative environments. The aim of this study is achieving similar results in a non-cooperative environment.

In order to motivate cooperation, we introduce a *monitoring mechanism*, whereby each node n monitors the sending rate of each of its neighbors. For each neighbor $\hat{n}$, n maintains *n.neighbor-_balance[$\hat{n}$]*, which is the difference between the number of data packets $\hat{n}$ has sent so far and the expected per-link throughput of $\frac{p}{k}$ data packets per-round. Note that, in a given round, $\hat{n}$ may have less than $\frac{p}{k}$ new data packets that have not yet been received at n, whereas in another round it may have more than $\frac{p}{k}$ data packets for n. Therefore, we allow for some slack in the balance. The allowed imbalance is captured by a negative threshold L. As long as $\hat{n}$'s balance with respect to n is greater than or equal to L, $\hat{n}$ is considered to be cooperative by n. But if $\hat{n}$'s balance with respect to n drops below L, then n terminates the connection with $\hat{n}$. Note that, as long as $\hat{n}$'s balance with respect to n is greater than or equal to L, the uploading rate from n to $\hat{n}$ is unaffected by the downloading rate from $\hat{n}$ to n. This independence is required in order to prove cooperation.

In order to further motivate nodes to adhere to the expected throughput, we introduce a per-link *penalty mechanism* that charges a neighbor with one additional *fine* packet for every round the neighbor has a negative balance with respect to the node, where a fine packet contains no useful data but has the same size as a data packet. If the node does not receive a fine packet from a neighbor with a negative balance, then it terminates its connection with that neighbor. Fine packets, as opposed to data packets, do not affect the node's balance. Therefore, a node is motivated to achieve a non-negative balance, where all sent packets contribute to its balance. Moreover, it is beneficial for nodes to have a strictly positive balance whenever possible. This is because there is no guarantee that a given neighbor will request at least $\frac{p}{k}$ packets from the node in forthcoming rounds. If a neighbor requests fewer than $\frac{p}{k}$ packets when the node's balance toward it is zero, then the balance becomes negative, and the node pays the fine. Each node chooses its maximal balance with respect to a given neighbor. This maximal balance is captured by the non-negative threshold H. Note that each node can set its own H parameter. As long as the node's balance with respect to a given neighbor $\hat{n}$ does not exceed H the node sends all the data packets that $\hat{n}$ requests from it, yet it refrains from sending data packets that would increase its balance with respect to $\hat{n}$ beyond H. Note that a node cannot optimize the value of H according to the session duration, as P is a random variable distributed exponentially. Note also that the penalty mechanism does not eliminate the need for L, since without this threshold, a selfish node could have sent only fine packets.

On the one hand, the allowed imbalance (the absolute value of L) should be large enough to reduce the probability of a cooperative node reaching L, in order to avoid overloading *S*. As we prove in Lemma 11, a node strives to achieve a non-negative balance with respect to all its neighbors, so that all the packets that it sends contribute to its balance. Therefore, since each round *S* sends new data packets to a random set of nodes, the number of new data packets a node sends to a given neighbor could be expected to Binomially distributed around a mean of the expected per-link throughput, i.e., $\frac{p}{k}$. Therefore, knowing the overall number of rounds and nodes, for a given value of L *S* can estimate the probability at which it is requested to send X data packets on behalf of nodes at a given round t.

On the other hand, a high imbalance allows a selfish node to receive many data packets, i.e., $|L|k$, without sending any data packets in return. Hence, there is an inherent tradeoff between the overhead incurred on *S* and the number of data packets a node can receive for free. For example, setting L to -200 is a good tradeoff between the two opposite requirements. On the one hand, if k=3, then a node can get only 600 data packets without contributing anything in return to the system. Since we assume that the multicast session is significantly longer, including at least $10{,}000$ packets, it seems like users will not be satisfied with getting a mere 600 packets and will therefore be motivated to contribute. On the other hand, such a bound is expected to incur a modest overhead on *S*. Note that a node cannot estimate the value of P from knowing the value of L, since it does not know tradeoff *S* chooses between the overhead incurred on it and the number of packets a selfish node can get for free. Therefore, a node's strategy does not depend on the value of L.

Finally, although nodes are motivated to have a non-negative balance, due to randomness, a node n may have an insufficient number of new packets for a given neighbor in order to be able to maintain a balance greater than or equal to L. In order to avoid a disconnection in such a scenario, n can ask *S* to send up to $\frac{p}{k}$ new data packets on behalf of it to a given neighbor $\hat{n}$ in return for sending the same number of fine packets to *S*. $\hat{n}$ counts *S*'s packets towards n's balance only if ignoring these packets would drop n's balance with respect to $\hat{n}$ below L. Hence, n contacts *S* only when its balance with respect to $\hat{n}$ drops below L. In addition, after the end of the multicast session, n can ask *S* to send to it up to $|L|k$ data packets in return for sending the same number of fine packets to *S*.

4.3 Detailed description

4.3.1 The source protocol

Each round, *S* creates p new data packets, and sends a copy of them to a random set of k nodes out of the N nodes. Upon receiving a request from a node n to send x data packets to another node $\hat{n}$, *S* verifies that:

Data structures:
neighbors – set of the overlay neighboring nodes.
my_balance[k] – outgoing balance, initially $\forall n \in neighbors, my\_balance[n] = 0$.
neighbor_balance[k] – incoming balance, initially $\forall n \in neighbors, neighbor\_balance[n] = 0$.
H – an upper bound on the balance, chosen by the node.
ids – set of data packet identifiers that the node has not yet received, initially $\emptyset$.
reqs[n] – a set of data packets identifiers to ask from neighbor n, initially $\forall n \in neighbors, req[n] = \emptyset$.
Parameters:
L – a lower bound on the balance (a negative number).

Figure 1: EquiCast's data structures and parameters.

(i) $x \leq \frac{p}{k}$; (ii) this request is followed by the sending of x fine packets from n; (iii) n and $\hat{n}$ are neighbors; (iv) neither n nor $\hat{n}$ has asked *S* to replace the other node with an emulated node; and (v) n is not pretending to be another node (IP-spoofing). The latter is checked, e.g., by sending a random string to n that n should send back to *S* in one of the fine packets. If n passes the checks, then *S* sends to $\hat{n}$ copies of x new data packets that it intends to distribute in the next round. If two or more of $\hat{n}$'s neighbors ask *S* to send data packets to $\hat{n}$, then *S* sends to $\hat{n}$ different packets on behalf of each neighbor. We neglect the possibility that in the next round $\hat{n}$ will be chosen by *S* to receive data packets from it, as the probability for this scenario is $k \cdot (\frac{k}{N})^2$.[2]

After the end of the multicast session, *S* provides a safety net for cooperative nodes that did not receive all the P multicast packets. Specifically, upon receiving x fine packets from a node n, *S* sends x data packets to n, for $x \leq |L|k$. In order to avoid server overloading in the end of the multicast session, we use the randomized back-off strategy described in [25].

4.3.2 The node protocol

Figure 1 presents the data structures and parameters maintained by an EquiCast node. The set *neighbors* holds the node's neighbors. The array *my_balance* holds the node's balance with respect to each of its neighbors, and the array *neighbor_balance* holds the neighbors' balances with respect to the node. The set ids contains identifiers of data packets that the node heard about (from one or more of its neighbors) but has not yet received. The array *reqs* holds identifiers of data packets that the node asks its neighbors to send to it. The (negative) threshold L determines the minimal allowed balance. Finally, each node chooses its own upper bound H on its balance with respect to a given neighbor, which defines its level of cooperation.

High-level illustration of the node's protocol is presented in Figure 2 and the pseudo-code of the node's protocol is presented is Figure 3. The node's protocol consists of four phases, which are executed sequen-

[2] In this case, if $\hat{n}$ is chosen by *S* to receive data packets in round t, then *S* can send data packets to $\hat{n}$ in round $t+1$.

tially.

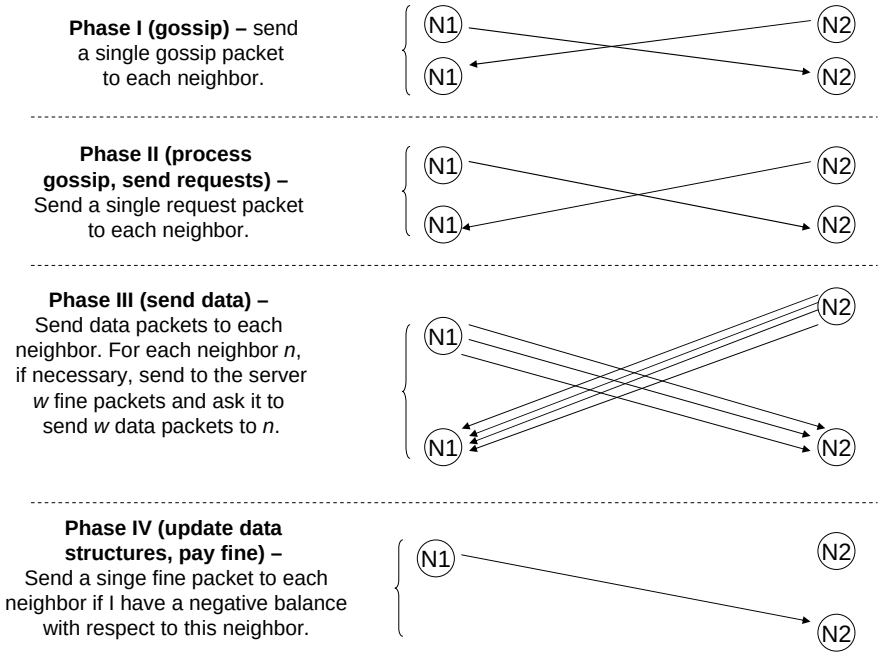

Figure 2: High-level illustration of a node's protocol with a given neighbor. Each node runs the very same protocol with each of its k overlay neighbors. The time is divided into rounds, and each round contains four phases, which are described in the above figure.

In the first phase, which lasts Δ time units, a node sends to its neighbors identifiers of data packets it received in the previous round (lines 1–6).

In the second phase, which also lasts Δ time units, if the node does not receive a gossip packet from some neighbor, then the node replaces its connection with that neighbor with a connection with an emulated node by calling to the procedure replace_neighbor (lines 7–9). Then, the node processes gossip packets it has received from its neighbors. For each identifier in ids, the set id_set holds all the neighbors that have the corresponding data packet. One such neighbor n is randomly chosen from this set, and the node asks n to send it the corresponding data packet by appending the identifier to $reqs[n]$.

In the third phase, which lasts $\delta - 3\Delta$ time units, if the node does not receive a request packet from some neighbor, then the node replaces its connection with that neighbor with a connection with an emulated node by calling to the procedure replace_neighbor (lines 19–21). Then, the node sends data packets to each of its

upon bootstrap: $neighbors \leftarrow$
identities of nodes received from *S*

Procedure replace_neighbor (node n)
$neighbors \leftarrow neighbors \setminus \{n\}$
contact *S* and ask for an emulated neighbor $\hat{n}$
$neighbors \leftarrow neighbors \bigcup \{\hat{n}\}$

Phase I (gossip)
1. */* Send gossip packets to neighbors */*
2. **foreach** $n \in neighbors$
3. create new gossip packet p with all the data packet identifiers received in the last round
4. send $\langle$GOSSIP,$p\rangle$ to n
5. **end foreach**
6. wait Δ time

Phase II (process gossip, send requests)
7. **foreach** $n \in neighbors$ from which no GOSSIP packet arrived
8. replace_neighbor (n)
9. **end foreach**
10. $ids \leftarrow$ set of identifiers received in gossip packets, whose corresponding data packets were not received yet
11. **foreach** $n \in neighbors$
12. $reqs[n] \leftarrow \emptyset$
13. **end foreach**
14. **foreach** $id \in ids$
15. $id\_set \leftarrow$ set of neighbors that gossiped about id
16. $ne \leftarrow$ a random neighbor from id_set so that $|reqs[ne]| < \frac{p}{k} + c - 3$
17. **if** there is no such ne **then continue**
18. $reqs[ne] \leftarrow reqs[ne] \cup \{id\}$
19. **end foreach**
20. **foreach** $n \in neighbors$
21. send $\langle$REQUEST,$reqs[n]\rangle$ to n
22. **end foreach**
23. wait Δ time

Phase III (send data)
19. **foreach** $n \in neighbors$ from which no REQUEST packet arrived
20. replace_neighbor (n)
21. **end foreach**
22. */* Send data packets */*
23. **foreach** $n \in neighbors$
24. send up to x data packets to n according to n's request, where $x = min(H + \frac{p}{k} - my\_balance[n],\ \frac{p}{k} + c - 3)$
25. $my\_balance[n] \leftarrow my\_balance[n] + x - \frac{p}{k}$
26. **if** $my\_balance[n] < L$ **then**
27. $w \leftarrow min(\frac{p}{k} + c - (x+3), \frac{p}{k})$
28. send *S* w fine packets and ask it to send w data packets to n
29. $my\_balance[n] \leftarrow my\_balance[n] + w$
30. **end foreach**
31. wait $\delta - 3\Delta$ time

Phase IV (update data structures, pay fine)
32. **foreach** $n \in neighbors$
33. $d \leftarrow$ number of data packets that I asked n to send me in phase II and were received from n in this round
34. **if** $neighbor\_balance[n] < L + \frac{p}{k}$ and I received in this round m data packets from *S* on behalf of n **then**
35. $d \leftarrow d + m$
36. $neighbor\_balance[n] \leftarrow neighbor\_balance[n] + d - \frac{p}{k}$
37. */* Send a fine packet (if needed) */*
38. **if** $my\_balance[n] < 0$ **then**
39. send a FINE packet to n
40. **end foreach**
41. wait Δ time
42. **foreach** $n \in neighbors$
43. */* Check if neighbor is OK */*
44. **if** $neighbor\_balance[n] < L$ or $neighbor\_balance[n] < 0$ and n did not send me a FINE packet in this round **then**
45. replace_neighbor (n)
46. **end foreach**

Figure 3: Code for EquiCast node.

neighbors[3]. Note that, according to the model (see Section 3), each node has an upload bandwidth of at most $p+kc$ packets per δ time units. Therefore, in the third phase, the node sends up to $x=\frac{p}{k}+c-3$ data packets to a given neighbor n, as long as its balance with respect to n does not exceed H (line 24). Additionally, the node increases its balance with respect to n by x. If the node's balance with respect to n is smaller than L, then the node asks *S* to send to n sufficiently many packets so that in the end of the current round the node will have a balance that is equal to or larger than L with respect to n (lines 26–28).

In the fourth phase, which lasts Δ time units, the node updates each neighbor's balance according to the number of data packets it received from the neighbor and from *S* on behalf of the neighbor in the previous phase (lines 32–36). Note that the node does not accept unsolicited data packets from its neighbors. Likewise, the node accepts data packets from *S* on behalf of some neighbor n only if, in the beginning of the fourth phase, n has a balance lower than $L+\frac{p}{k}$ with respect to the node. Then, if the node has a negative balance with respect to n, it sends one fine packet to n. Finally, if n either has a balance lower than L or did not send the fine packet it was required to, then the node terminates its connection with n.

5 Proof of Cooperation

Recall that P, the number of data packets in a session, is a random variable distributed exponentially with a large expectation, at least an order of magnitude larger than $|L|k$. Hence, in every round, *S* is expected to create more than $|L|k$ new data packets in the future. In this section, we neglect the probability that, starting from some round t, *S* will create less than $|L|k$ new data packets, and hence we assume that, in every round, the probability that *S* will create more than $|L|k$ data packets in the future is 1. Moreover, for every constant $const$, $\frac{const}{R}$ is negligible (recall that $R = \frac{P}{p}$ is the total number of rounds in the multicast session), and for simplicity is assumed to be 0.

We say that a node n *maintains* a connection with another node $\hat{n}$ in some phase z of some round t if, in phase z of round t, n runs the protocol's code (described in Figure 3) with respect to $\hat{n}$ without changing any of the protocol's parameters except H. Note that $\hat{n}$ can be either a real node or a node emulated by *S*. We say that n maintains a connection with $\hat{n}$ throughout the multicast session if n maintains a connection with $\hat{n}$ in every phase of each of the first R rounds of the multicast session (i.e., in every round in which *S*

[3]We assume that a node does not send an arbitrary (dummy) packet instead of sending a genuine data packet, since this problem can be easily solved by the server providing some fingerprint of each packet, which the receiver node can check.

creates new data packets).

Throughout this section, we use the following notations related to a node n and a given neighbor $\hat{n}$ of n: $b_t(n,\hat{n})$ is n's balance towards $\hat{n}$ after t rounds as stored in $n.my\_balance[\hat{n}]$. $x_t(n,\hat{n})$ is the number of data packets n (or *S* on behalf of n) sends to $\hat{n}$ during round t, and $X_t(n,\hat{n}) = \sum_{i=0}^{i=t} x_t(n,\hat{n})$.

In Section 5.1, we prove several basic (technical) properties of the protocol. In Section 5.2, we define the set of *protocol-obedient strategies (POSs)*, and we prove that every POS in which a node cooperates with all its neighbors strictly dominates every POS in which it does not. In addition, we prove that if all the nodes choose POSs, then each node receives all the multicast packets. In Section 5.3, we prove that if all the nodes choose POSs, then no node can unilaterally reduce its cost by changing its strategy to a non-POS. In addition, we prove that, in this case, all the nodes receive all the multicast packets. Finally, in Section 5.4, we prove that each node chooses a non-negative H parameter.

5.1 Basic properties

Lemma 1. *For every two neighboring nodes n and $\hat{n}$, if, starting from the initialization of the connection between them, both n and $\hat{n}$ maintain the connection between them in every phase of the first t rounds, then $n.my\_balance[\hat{n}] = \hat{n}.neighbor\_balance[n]$ in the end of round t', for every $t' \leq t$.*

Proof. By induction on the round number.

Base: There are two cases. In the first case, both n and $\hat{n}$ are not emulated nodes. In this case, the connection between the two nodes is initialized in the beginning of the multicast session (i.e., in the end of round "0"), and, upon the initialization of the connection, $n.my\_balance[\hat{n}] = \hat{n}.neighbor\_balance[n] = 0$. In the second case, either n or $\hat{n}$ is an emulated node. Without loss of generality, assume that $\hat{n}$ is an emulated node, and that n receives the identity of $\hat{n}$ from *S* during round r, for some $r \geq 0$. In this case, the connection between the two nodes is initialized in round r, and $n.my\_balance[\hat{n}] = \hat{n}.neighbor\_balance[n] = L$ upon the initialization of the connection. We note that, in round r, n and $\hat{n}$ do not send data packets to each other; this is since the connection is initialized after the end of the first (gossip) phase of round r, and hence, in round r, no gossip packets are sent on the connection between these two nodes, and therefore, in this round, no data packets are sent on this connection either. Therefore, in the end of round r, $n.my\_balance[\hat{n}] = \hat{n}.neighbor\_balance[n] = L$.

Step: Assume that, in the end of round t, $n.my\_balance[\hat{n}] = \hat{n}.neighbor\_balance[n]$. We will prove that,

in the end of round $t+1$, $n.my\_balance[\hat{n}] = \hat{n}.neighbor\_balance[n]$.

In round $t+1$, both $n.my\_balance[\hat{n}]$ and $\hat{n}.neighbor\_balance[n]$ are reduced by $\frac{p}{k}$ (see Figure 3, lines 25 and 36). In addition, in round $t+1$, $n.my\_balance[\hat{n}]$ and $\hat{n}.neighbor\_balance[n]$ are increased upon the sending of data packets from n and from *S* on behalf of n to $\hat{n}$ (see Figure 3, lines 25, 29, and 36).

When n sends x new data packets to $\hat{n}$, $n.my\_balance[\hat{n}]$ is increased by x (see Figure 3, line 25). Since there is no packet loss, these packets are received at $\hat{n}$. We note that n sends to $\hat{n}$ only data packets that $\hat{n}$ requested from it in phase II of round $t+1$; this is since $\hat{n}$ ignores unsolicited data packets, as it maintains the connection with n (see Figure 3, line 33). Therefore, upon receiving the x data packets, $\hat{n}$ increases $\hat{n}.neighbor\_balance[n]$ by x.

If $n.my\_balance[\hat{n}]$ drops below L during phase III of round $t+1$, then n sends w fine packets to *S* and it asks *S* to send w data packets on behalf of it to $\hat{n}$ (see Figure 3, lines 26–28). Additionally, $n.my\_balance[\hat{n}]$ is increased by w (see Figure 3, line 29). We note that n does not request from *S* to send to $\hat{n}$ more than $\frac{p}{k}$ data packets, as *S* ignores such requests (see Section 4.3). Hence, upon receiving the request from n, *S* sends w data packets on behalf of n to $\hat{n}$. Since there is no packet loss, these packets are received at $\hat{n}$. According to the induction assumption and since n and $\hat{n}$ maintain the connection between them, $\hat{n}.neighbor\_balance[n] < L + \frac{p}{k}$ in the beginning of phase IV of round $t+1$. Hence, $\hat{n}$ accepts the data packets received from *S*, and it increases $\hat{n}.neighbor\_balance[n]$ by w (see Figure 3, lines 34–36). Finally, we note that if n asks *S* to send data packets to $\hat{n}$ when $n.my\_balance[\hat{n}] \geq L$, then $\hat{n}$ ignores these data packets (see Figure 3, line 33), since in this case $\hat{n}.neighbor\_balance[n] \geq L + \frac{p}{k}$, and hence n asks *S* to send data packets to $\hat{n}$ only if $n.my\_balance[\hat{n}] < L$ during phase III of a given round. □

Lemma 2. *For every two neighboring nodes n and $\hat{n}$, if, starting from the initialization of the connection between them, both n and $\hat{n}$ maintain the connection between them in every phase of the first t rounds, then this connection is not terminated in the first t rounds.*

Proof. Without loss of generality, we will prove that $\hat{n}$ does not terminate the connection with n (i.e., $\hat{n}$ does not call to the procedure replace_neighbor with n's identity) in round t', for every $t' \leq t$. Since $\hat{n}$ maintains the connection with n, then it terminates the connection with n in round t' only if one (or more) of the following situations occurs: (i) it does not receive a gossip packet from n in phase I of round t'; or if (ii) it does not receive a request packet from n in phase II of round t'; or if (iii) either $\hat{n}$ is an emulated node or, in the end of round t', $\hat{n}.neighbor\_balance[n] < 0$, and $\hat{n}$ does not receive a fine packet from n in round

t'; or if (iv) in the end of round t', $\hat{n}.neighbor\_balance[n]<L$.

Since n maintains the connection with n, then (i), (ii), and (iii) do not happen. In addition, we note that n can ensure that, in the end of each round, $n.my\_balance[\hat{n}] \geq L$ by asking S to send data packets to $\hat{n}$ when $n.my\_balance[\hat{n}]<L$ (during phase III of a given round). Hence, according to Lemma 1, (iv) does not happen either. □

Lemma 3. *If a node n maintains a connection with another node $\hat{n}$ through the first t rounds of the multicast session, then $X_t(n,\hat{n})=\frac{tp}{k}+b_t(n,\hat{n})$.*

Proof. By induction.

Base: t = 0. $X_0(n,\hat{n}) = b_0(n,\hat{n}) = 0$. Therefore, $X_0(n,\hat{n}) = \frac{tp}{k} + b_0(n,\hat{n})$.

Step: Assume $X_t(n,\hat{n}) = \frac{tp}{k} + b_t(n,\hat{n})$. We will prove that $X_{t+1}(n,\hat{n}) = \frac{(t+1)p}{k} + b_{t+1}(n,\hat{n})$.

$X_{t+1}(n,\hat{n}) = X_t(n,\hat{n}) + x_{t+1}(n,\hat{n}) = \frac{tp}{k} + b_t(n,\hat{n})+x_{t+1}(n,\hat{n})$. Since n maintains the connection with $\hat{n}$, we know that $b_{t+1}(n,\hat{n}) = b_t(n,\hat{n}) + x_{t+1}(n,\hat{n}) - \frac{p}{k}$ (see Figure 3, lines 25 and 29). Therefore, $b_t(n,\hat{n}) + x_{t+1}(n,\hat{n}) = b_{t+1}(n,\hat{n}) + \frac{p}{k}$. Hence, $X_{t+1}(n,\hat{n}) = \frac{(t+1)p}{k} + b_{t+1}(n,\hat{n})$. □

Lemma 4. *If k neighbors of a node n maintain a connection with it throughout the multicast session, then n receives from its neighbors and from S on behalf of its neighbors at least $P+Lk$ data packets*[4].

Proof. By Lemma 3, for every neighbor $\hat{n}$ of n, $X_R(\hat{n},n) = \frac{Rp}{k} + b_R(\hat{n},n)$. Recall that $b_R(n,\hat{n}) \geq L$ and $R = \frac{P}{p}$. Hence, from all its k neighbors, n receives at least $Rp + Lk = P + Lk$ data packets. □

Lemma 5. *The per-round overhead of maintaining a connection over the entire multicast session is at least $\frac{p}{k}+2$ packets and at most $\frac{p}{k}+c$ packets.*

Proof. The overhead incurred on a node n for maintaining a connection with another node $\hat{n}$ consists of: (i) data overhead (X_R), i.e., packets that contribute to n's balance with respect to $\hat{n}$, (ii) gossip/request packets, and (iii) penalty packets.

The maximum data overhead incurred by maintaining the connection with $\hat{n}$ is $\frac{p}{k} + c - 3$ data packets per-round (see Figure 3, lines 24–29). By Lemma 3, and since L is fixed, $\frac{X_R(n,\hat{n})}{R} = \frac{p}{k} + \frac{b_R(n,\hat{n})}{R} \geq \frac{p}{k} + \frac{L}{R} = \frac{p}{k}$. The gossip/request overhead is fixed, namely: two packets per-round. The penalty on either a negative balance or on maintaining a connection with an emulated node is one fine packet per round,

[4]Recall that L is negative.

and zero otherwise. Hence, the minimal and maximal per-round overheads are $\frac{p}{k}+2$ and $\frac{p}{k}+c$ packets, respectively. □

Lemma 6. *If a node n maintains connections with at most k-1 nodes throughout the multicast session and in addition, it communicates with a bounded number of nodes throughout a bounded number of rounds, then $f_n=\infty$.*

Proof. We first note that, during the multicast session, n cannot request from *S* to send it data packets. From at most $k-1$ neighbors with which n maintains connections throughout the multicast session (and from *S* on behalf of these neighbors), n can receive at most $x=(k-1)(\frac{p}{k}+c-3)$ data packets per round. Recall that $(k^2-k)(c-3)<p$. Hence, $x<p$. We note that $x<p$ even if n communicates with an additional bounded number of nodes throughout a bounded number of rounds. This is since n receives a bounded number, denoted as num, of data packets from these nodes, and hence $\frac{xR+num}{R}=x<p$. Finally, if n receives up to $|L|k$ data packets from *S* after the end of the multicast session, then it still cannot receive all the P multicast packets, since $\frac{xR+|L|k}{R}=x<p$. Hence, $f_n=\infty$. □

Lemma 7. *If a node n maintains connections with k nodes that also maintain a connection with it throughout the multicast session, then $f_n<\infty$ and $\frac{f_n}{R}\leq p+kc$.*

Proof. By Lemma 4, if n maintains connections with k nodes that also maintain connections with it throughout the multicast session, then n receives at least $P+Lk$ data packets from its neighbors and from *S* on behalf of its neighbors. In addition, after the end of the multicast session, n can receive up to $|L|k$ data packets from *S* (in return for sending *S* a fine packet for each data packet), and hence $f_n=s_n<\infty$.

By Lemma 5, maintaining k connections incurs sending at most $p+kc$ packets per-round. In addition, since $\frac{|L|k}{R}=0$ (L and k are fixed), sending at most $|L|k$ fine packets in the end of the multicast session does not increase the per-round overhead. Thus, $\frac{f_n}{R}\leq p+kc$. □

We now discuss the case in which a node n maintains connections with more than k nodes throughout the multicast session. Note that by Lemma 5 and due to bandwidth limitations, n cannot maintain more than $\lfloor\frac{p+kc}{\frac{p}{k}+2}\rfloor$ connections. Below, we prove that maintaining connections with $k+1$ or more nodes incurs a higher cost than maintaining connections with k nodes.

Lemma 8. *Every strategy in which a node n exclusively maintains connections with k nodes (i.e., n communicates only with these k nodes) throughout the multicast session incurs a lower cost than every strategy in which n maintains connections with j nodes throughout the multicast session, where $j>k$.*

Proof. By Lemma 5, if n maintains connections with $k+1$ or more nodes throughout the multicast session, then $\frac{s_n}{R} \geq (k+1)(\frac{p}{k}+2)$, i.e., $\frac{f_n}{R} \geq (k+1)(\frac{p}{k}+2)$. By Lemma 7, if n exclusively maintains connections with k nodes throughout the multicast session, then $\frac{f_n}{R} \leq p+kc$. Recall that $k^2(c-2)-2k<p$. Hence, $p+kc<(k+1)(\frac{p}{k}+2)$. □

5.2 The set of Protocol-Obedient Strategies (POSs)

We now define the set of POSs. Roughly speaking, a node that chooses a POS can choose which connections to maintain among those allowed by the protocol, and it doesn't communicate with anyone with which it does not maintain a connection. In this section, we prove that, if all nodes choose dominant POSs, then each node maintains connections with its initial k neighbors throughout the entire multicast session and it receives all the multicast packets.

Definition 1 (Protocol-obedient strategy (POS))**.** *A node's strategy is protocol-obedient if:*

[POS 1.] *In the beginning of the multicast session, n chooses some subset of the initial k nodes given to it by S to be connected to.*

[POS 2.] *In the beginning of every phase of every round, n chooses for each node that it is connected to whether to disconnect from it or to remain connected to it, and moreover, n chooses whether to ask S for any number of new emulated nodes to connect to as long as n does not maintain connections with more than k emulated nodes.*

[POS 3.] *For each node $\hat{n}$, n communicates with $\hat{n}$ in some phase z of some round t if and only if n is connected to $\hat{n}$ in phase z of round t according to the choices above, and moreover, in this case, n maintains the connection with $\hat{n}$ in phase z of round t.*

[POS 4.] *For any node $\hat{n}$, if $\hat{n}$ does not maintain the connection with n in phase z of round t, then n terminates its connection with $\hat{n}$ (according to the protocol) in phase $z+1$ of round t, and it does not further communicate with $\hat{n}$ starting from this phase.*

Note that, in particular, following the protocol (see Figure 3) is a POS.

We believe that it is reasonable to assume that most users will run POSs, since the typical user either does not have the technical knowledge to modify an application code or he/she does not download a hacked code as there is a risk in downloading such a code. In addition, in many P2P applications, a node communicates with nodes whose identities are received from a centralized server. For example, in BitTorrent, a node locates other nodes by contacting a "tracker", which is a centralized process that keeps track of all nodes interested in a specific file [6, 13]. Moreover, in the next section, we prove that for each node n, if all the nodes that n communicates with choose POSs, then n also chooses a POS. That is, hacking the protocol's code cannot reduce n's cost if neither at least one of n's initial neighbors also hacked the protocol's code nor n succeeds to locate by itself identities of nodes that also hacked the protocol's code.

Definition 2 (k-protocol-obedient strategy (k-POS))**.** *A POS in which a node maintains exactly k connections throughout the entire multicast session is called a k-POS.*

We note that a k-POS is always feasible, since a node can always maintain connections with k emulated nodes that will also maintain connections with it. The following lemma shows that a k-POS is a dominant POS.

Lemma 9. *A k-POS strictly dominates every POS in which n maintains connections with j nodes, where $j \neq k$.*

Proof. We first note that n can communicate only with either the initial k neighbors given to it by *S* or with up to k emulated nodes; this is since n chooses a POS. We also note that maintaining a connection for a bounded number of rounds cannot reduce n's cost, since from this connection n receives a bounded number of data packets, denoted as num, and $\frac{num}{R}=0$. Hence, the lemma follows from Lemmas 6, 7, and 8. □

We next show that, if all nodes choose POSs, then a node benefits more from connections with its original k neighbors than from connections with emulated ones.

Lemma 10. *Assume that (i) all nodes choose POSs; (ii) a node n maintains a connection with a non-emulated node $\hat{n}$ in some phase z of some round t; and (iii) $\hat{n}$ also maintains a connection with n in phase z of round t. Then, n does not replace its connection with $\hat{n}$ with a connection with an emulated node e.*

Proof. Recall that e's interface is identical to $\hat{n}$'s interface with the following two exceptions: i) n's balance with respect to e is initialized to the lowest possible balance, i.e., L; and ii) in each round, n must send a fine

packet to e, regardless of its balance with respect to e, otherwise e terminates its connection with n. Hence, there is no difference between the data receiving rate from $\hat{n}$ and the data receiving rate from e.

The overhead of maintaining a connection with either $\hat{n}$ or e is composed of: (i) data overhead, (ii) gossip/request packets, and (iii) penalty packets. The gossip/request overhead is fixed. The data sending rate to e is larger than or equal to the data sending rate to $\hat{n}$, since n's balance with respect to e is initialized to the lowest possible balance, i.e., L. The penalty overhead incurred by maintaining a connection with e is larger than or equal to the penalty overhead incurred by maintaining a connection with $\hat{n}$, since, at each round, n is required to send a penalty packet to e, regardless of its balance with respect to e. Finally, in order to maintain a connection with e, n needs to send a join message to *S*. Hence, the overhead of maintaining a connection with e is larger than the overhead of maintaining a connection with $\hat{n}$. Therefore, since there is no difference between the data receiving rate from $\hat{n}$ and the data receiving rate from e, n does not replace its connection with $\hat{n}$ with a connection with e. □

Theorem 1. *If all nodes choose strongly dominant strategies out of the set of POSs, then every node n exclusively maintains connections with its initial k neighbors throughout the multicast session, and it receives all the multicast packets.*

Proof. By Lemmas 9 and 10, n's strategy is to exclusively maintain connections with its initial k neighbors throughout the multicast session. By Lemma 2, these connections are maintained. Hence, n exclusively maintains connections with its initial k neighbors throughout the multicast session. Finally, by Lemma 7, n receives all the multicast packets. □

5.3 Unilateral defection from the protocol

In this section, we prove that if all the nodes, except for one node n, choose a strategy out of the set of possible POSs, then n's cost is minimized by choosing a k-POS. In other words, if all the nodes choose POSs, then no node can reduce its cost by unilaterally changing its strategy to a non-POS. Furthermore, we show that, in this case, all the nodes receive all the multicast packets.

Theorem 2. *If all the nodes, except for one (rational) node n, choose a strategy out of the set of possible POSs, then n also chooses a POS.*

Proof. We shall prove that n complies with statements POS 1– POS 4, which together comprise the definition of a POS (see Section 5.2).

We first note that each node, including n, complies with POS 1, since, in the beginning of the multicast session, each node receives from *S* identities of k nodes, and each node can choose whether to connect to each of its initial k neighbors.

Throughout the entire multicast session, n benefits nothing from sending packets to nodes whose identities were not received from *S*, since these nodes send no packets to n; this is since these nodes choose POSs which prohibit communication with nodes whose identities were not received from *S*. In addition, we note that if a connection between n and one of its neighbors $\hat{n}$ is terminated, then $\hat{n}$ refuses to communicate (i.e., to send packets) with n, since $\hat{n}$ chooses a POS which replaces a neighbor that does not maintain a connection with an emulated neighbor. Hence, starting from the second round, n adds connections to emulated nodes only. Finally, we note that n cannot be connected to more than k emulated nodes, since *S* does not allow such a case. Therefore, n complies with POS 2.

We note that n does not communicate with any node $\hat{n}$ is some phase z of some round t if it does not maintain a connection with $\hat{n}$ in this phase; this is since $\hat{n}$ chooses a POS that dictates terminating the connection with n in phase $z+1$ of round t if n does not maintain the connection with $\hat{n}$ in phase z of round t, and therefore n benefits nothing from communication with $\hat{n}$ in phase z of round t if this communication is not according to the protocol. Hence, n complies with POS 3.

Since n is rational, we note that n terminates a connection with a neighbor $\hat{n}$ if $\hat{n}$ does not maintain the connection with n; this is since $\hat{n}$ chooses a POS, and hence either $\hat{n}$ maintains the connection with n or it does not send any packets to n, and hence n benefits nothing from maintaining a connection with $\hat{n}$ if $\hat{n}$ does not maintain a connection with n. Hence, n complies with POS 4. □

Note that Theorem 2 holds even if all the nodes, except for n, are not rational.

Next, we establish that hacking the protocol's code cannot reduce a node's cost if neither at least one of the node's initial neighbors also hacked the protocol's code nor the node's succeeds to locate by itself identities of nodes that also hacked the protocol's code.

Theorem 3. *If all of a node's n's initial k neighbors are rational and choose POSs and n cannot locate an identity of a node that does not choose a POS, then n exclusively maintains connections with its initial k neighbors throughout the multicast session, and it receives all the multicast packets.*

Proof. The theorem follows directly from Theorems 1 and 2. □

5.4 Choosing H

Next, we prove that each node chooses a non-negative H parameter.

Lemma 11. *Assume that a node n maintains connections with k nodes throughout the multicast session. Assume also that some neighbor $\hat{n}$ of n requests from n to send to it $q \leq \frac{p}{k}+c-3$ data packets in some round r, and in the beginning of round r, n has a negative balance of b with respect to $\hat{n}$. Then, in round r, n sends $min(|b|, q)$ data packets to $\hat{n}$.*

Proof. We first note that, in the end of each round t, $b_t(n, \hat{n}) \geq L$, since n maintains the connection with $\hat{n}$. Thus, the sending rate to $\hat{n}$ does not affect the data receiving rate from $\hat{n}$, and hence n can minimize its sending rate to $\hat{n}$ in order to minimize its cost.

The per-round overhead incurred by maintaining the connection with $\hat{n}$ consists of: (i) data overhead ($\frac{X_R}{R}$), (ii) gossip/request packets, and (iii) penalty packets. The gossip/request overhead is fixed. Hence, n tries to minimize the data and penalty overheads.

By Lemma 3, $\frac{X_R(n,\hat{n})}{R} = \frac{p}{k} + \frac{b_R(n,\hat{n})}{R}$. The per-round data overhead is bounded from below by $\frac{p}{k} + \frac{L}{R}$. Since L is a constant that does not depend on R, we can neglect $\frac{L}{R}$, i.e., assume it is zero. The per-round penalty overhead is the percentage of rounds in which the balance is negative. Recall that, in each round, the probability that *S* will create more that $|L|k$ data packets in the future is 1. Hence, the overall cost is lower if n maintains a zero balance with respect to $\hat{n}$ in the end of each round when this is possible. Therefore, n sends $min(|b|, q)$ data packets to $\hat{n}$ in round r. □

6 Dynamic Setting

We now describe in a nutshell a dynamic version of EquiCast, called *DEC (Dynamic EquiCast)*, in which nodes can join and leave the protocol during its execution. However, we assume that the churn rate (i.e., the rate at which nodes join and leave the system) is low. In Section 6.1 we describe a version of DEC that is suitable for a live streaming setting, in which a user expects to receive only new data packets that are disseminated after its joining time, and in Section 6.2 we describe a version of DEC that is suitable for content distribution, e.g., downloading a file, in which a user expects to get all the P data packets. These

two versions of DEC are very similar, and the major difference between these two versions is the definition of the cost function. Below, we detail only the differences between EquiCast and DEC. DEC's proof of cooperation can be trivially derived from EquiCast's proof of cooperation, i.e., all the proofs of Section 5 are either also true for a dynamic setting or can be easily revised for such a setting.

6.1 DEC for a live streaming setting

Architecture

DEC is deployed on top of a dynamic overlay that supports node joins and leaves in which each node in the overlay has exactly k neighbors for some parameter k (property KRRG1). Recall that only property KRRG1 is required in order to prove EquiCast's/DEC's cooperation enforcement scheme. For example, DEC can use the overlay in [19], which is a dynamically maintained k-regular graph composed of $\frac{k}{2}$ Hamiltonian cycles. In such an overlay, each leave and join operation entails the sending of only $O(\frac{k}{2})$ and $O(\frac{k}{2} \cdot \log_{\frac{k}{2}} N)$ messages, respectively. Therefore, such a dynamic overlay can be efficiently maintained even in a highly dynamic setting.

The cost function

Recall that in EquiCast the cost function for a node n_i is defined as:

$$f_i(st_0, st_1, ..., st_N) = \begin{cases} \infty & \text{if } r_i < P \\ s_i & \text{if } r_i = P, \end{cases}$$

where P is the overall number of data packets disseminated by *S*, r_i/s_i is the total number of data packets received/sent by n_i throughout the multicast session, and st_0–st_N are *S*'s and the nodes strategies. That is, if n_i receives all the multicast packets, then its cost is the number of packets it sends during the multicast session. Otherwise, n_i's cost is infinite.

In the static case each node can receive at least $P - |L|k$ data packets from its neighbors/*S* during the multicast session and up to $x \leq |L|k$ data packets from *S* after the end of the multicast session at the cost of sending *S* x packets. In a dynamic setting a node can join the multicast session after multicast session has started. Specifically, in the worst case, a node that is connected to the overlay for m rounds can receive only $max(mp - |L|k, 0)$ data packets, where the multicast rate is p data packets per round. In a

dynamic live streaming setting, DEC's cost function is obtained from EquiCasts cost function by replacing the requirement to receive all the P multicast packets with the requirement to receive $m{\cdot}p$ data packets, where m is the number of rounds during which the node is connected to the overlay. That is, if n_i receives all the multicast packets that are created starting from its joining time, then its cost is the number of packets it sends during the multicast session. Otherwise, n_i's cost is infinite. Formally, DEC's cost function for a dynamic live streaming setting is defined as:

$$f_i(st_0, st_1, ..., st_N) = \begin{cases} \infty & \text{if } r_i < mp \\ s_i & \text{if } r_i = mp, \end{cases}$$

A join operation

A joining node n sends a *join* message to *S*. Upon receiving this request, *S* incorporates n into the overlay, e.g., by inserting n between $\frac{k}{2}$ pairs of neighboring nodes [19]. For example, assume that nodes n_1 and n_2 are connected to the overlay prior to n's joining, and n becomes n_1's neighbor instead of n_2. We describe how *S* sets n's and n_1's incoming ($neighbor\_balance$) and outgoing ($my\_balance$) balances with respect to each other.

Prior to incorporating n into the overlay, *S* asks both n_1 and n_2 for their incoming and outgoing balances with respect to each other. If these balances do not match, then *S* disconnects both n_1 and n_2 from the overlay by sending an appropriate message to all their neighbors. Hence, since both n_1 and n_2 are rational, they could be expected to correctly report about their incoming and outgoing balances with respect to each other.

Denote n_1's outgoing and incoming balances with respect to n_2 in the end of round t as B_{12} and B_{21}, respectively. We would like to ensure that n_1's cost will not increase due to n's joining. Therefore, in the beginning of round $t{+}1$, both n_1's outgoing balance with respect to n and n's incoming balance with respect to n_1 are set to B_{12}. Additionally, in the beginning of round $t{+}1$, both n_1's incoming balance with respect to n and n's outgoing balance with respect to n_1 are set to $max(B_{21}, 0)$. This is to ensure that n will not pay a fine for n_2's negative outgoing balance with respect to n_1. Finally, if $B_{21}{<}0$, then *S* sends $|B_{21}|$ new data packets to n_1, in order to ensure that it receives at least $m{\cdot}p{-}|L|k$ data packets during the m rounds it is connected to the overlay. Similarly, if $B_{12}{>}0$, then *S* sends B_{12} new data packets to n.

A leave operation

A leaving node n sends a *leave* message to *S*. Upon receiving this request, *S* removes n from the overlay, e.g., by connecting each pair of n's neighbors with each other [19]. For example, assume that prior to n's leaving n was connected to nodes n_1 and n_2, and n_1 and n_2 become neighbors after n's leaving. We describe how *S* sets n_1's and n_2's incoming and outgoing balances with respect to each other.

Prior to leaving the overlay, n sends to *S* its incoming and outgoing balances with respect to both n_1 and n_2. Note that n cannot gain anything from reporting about false balances, and hence n could be expected to correctly report about its balances with respect to n_1 and n_2.

Denote n_1's and n_2's outgoing balances with respect to n in the end of round t as B_{1n} and B_{2n}, respectively. We would like to ensure that n_1's and n_2's cost will not increase due to n's leaving. Therefore, in the beginning of round $t{+}1$, n_1's and n_2's outgoing balances with respect to each other are set to B_{1n} and B_{2n}, respectively. Additionally, in order to ensure the protocol's correctness, in the beginning of round $t{+}1$, n_1's and n_2's incoming balances with respect to each other are set to B_{2n} and B_{1n}, respectively.

Denote n_1's and n_2's incoming balances with respect to n in the end of round t as B_{n1} and B_{n2}, respectively. If $B_{2n}{>}B_{n1}$, then n_1 may not receive $m{\cdot}p{-}|L|k$ data packets during the m it is connected to the overlay. Hence, in such a case, *S* sends $B_{2n}{-}B_{n1}$ new data packets to n_1. Similarly, if $B_{1n}{>}B_{n2}$, then *S* sends $B_{1n}{-}B_{n2}$ new data packets to n_2.

6.2 DEC for a content distribution setting

In a content distribution setting, a user expects to receive all the file segments regardless of its joining time. Therefore, in such a setting DEC's cost function is identical to EquiCast's cost function. That is a node's cost is finite only if it gets all the P data packets. If the node arrival rate is sufficient to replace departing nodes, e.g., as assumed in [6, 15], then *S* can repeatedly multicast all the P data packets in cycles. This way, similarly as in a static setting, a joining node is able to get all the P data packets while staying connected to the overlay for exactly the duration of the multicast session, i.e., $\lceil \frac{P}{p} \rceil$ rounds. Under these assumptions, there is no extra overhead on *S*.

If the above assumptions do not exist, then as proven in [15] self-downloading is not feasible (Herman and Johnen [15] proved that in a dynamic setting self-downloading is feasible only if there is a stream of nodes that arrive, share while downloading, and exit when finished, provided the arrival rate is sufficient to

replace departing nodes, and the download bandwidth is the is the same for all nodes). However, in such a setting, DEC can support a hybrid downloading scheme in which nodes are motivated to self-download from each other, i.e., join to the overlay before the start of the multicast session, though a node that joins to the overlay after the start of the multicast session and does not get all the P data packets from its neighbors can receive missing data packets from *S* at a substantial higher cost compared to self-downloading. For example, *S* can require a node to send it $(kc + \frac{|L|k}{\lceil \frac{P}{p} \rceil})x$ data packets in order to receive x data packets from *S* (As we prove in Lemma 5, the per-round overhead of maintaining a connection over the entire multicast session is at least $\frac{p}{k}+2$ packets and at most $\frac{p}{k}+c$ packets). On the one hand, this payment mechanism encourages a node to join the overlay prior to the start of the multicast session (and by that reducing the overhead incurred on *S*), since such an early joining minimizes the node's cost (see Lemma 5). On the other hand, such a payment mechanism allows each node to retrieve all the P multicast packets, i.e., achieving a finite cost, regardless to its joining time to the overlay. Finally, one can avoid the server overloading in the end of the multicast session by using the randomized back-off strategy described in [25].

7 Conclusions

Freeloaders degrade the performance of P2P systems and may lead to their collapse. We have tackled the problem of freeloaders in a P2P multicast protocol from a theoretic perspective by modeling the system as a non-cooperative game. We have introduced EquiCast, a P2P multicast protocol for selfish environments. In such environments, EquiCast distributes all the multicast packets to all the nodes. We have formally proven EquiCast's cooperation enforcement scheme, namely: in EquiCast, for each node, collaborating with all its neighbors is a strongly dominant strategy. We are unaware of any previous P2P multicast protocol that has been shown to enforce cooperation in environments in which all the nodes are selfish. We have also proven that EquiCast incurs a constant load on each node, and hence it can support large groups of users. Finally, we have described a dynamic version of EquiCast, which supports node joins and leaves.

Acknowledgements: We thank Amir Ronen and Vadim Drabkin for many helpful comments.

References

[1] EMULE-PROJECT.NET. eMule site. http://www.emule-project.net/.

[2] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth. Bar fault tolerance for cooperative services. In *20th ACM Symposium on Operating Systems Principles*, 2005.

[3] A. Blanc, Y.-K. Liu, and A. Vahdat. Designing incentives for peer-to-peer routing. In *Proceedings of the IEEE Infocom Conference*, 2005.

[4] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, October 2003.

[5] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *ACM SIGCOMM*, August 2003.

[6] B. Cohen. Incentives build robustness in BitTorrent. In *1st Workshop on the Economics of Peer-to-Peer Systems*, 2003.

[7] L. Cox and B. Noble. Samsara: Honor among thieves in peer-to-peer storage. In *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, 2003.

[8] J. Feigenbaum, C. H. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63(1):21–41, 2001.

[9] J. Friedman. On the second eigenvalue and random walks in random d-regular graphs. Combinatorica, vol. 11, pp. 331-362, 1991.

[10] D. Fudenberg and J. Tirole. *Game Theory*. The MIT Press, 1991.

[11] C. Gkantsidis and P. R. Rodriguez. Network coding for large scale content distribution. In *Proceedings of the IEEE Infocom Conference*, 2005.

[12] A. Habib and J. Chuang. Incentive mechanism for peer-to-peer media streaming. In *International Workshop on Quality of Service (IWQoS '04)*, 2004.

[13] D. Hales and S. Patarin. How to cheat bittorrent and why nobody does. TR UBLCS-2005-12, Department of Computer Science University of Bologna, May 2005.

[14] M. Haridasan and R. van Renesse. Defense against intrusion in a live streaming multicast system. In *IEEE International Conference on Peer-to-Peer Computing*, 2006.

[15] T. HERMAN and JOHNEN. Strategies for peer-to-peer downloading. In *Inform. Process. Lett. 94, 5, 203209*, 2005.

[16] T. Karagiannis, P. Rodriguez, and D. Papagiannaki. Should isps fear peer-assisted content distribution? In *ACM USENIX IMC*, 2005.

[17] M. Kim and M. Medard. Robustness in large-scale random networks. In *Proceedings of the IEEE Infocom Conference*, 2004.

[18] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, October 2003.

[19] C. Law and K. Siu. Distributed construction of random expander networks. In *Infocom*, 2003.

[20] D. Levin, R. Sherwood, and B. Bhattacharjee. Fair File Swarming with FOX. In *5th Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[21] H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR Gossip. In *Proceedings of the 7th Symposium on Operating System Design and Implementation (OSDI '06)*, 2006.

[22] R. Melamed and I. Keidar. Araneola: A scalable reliable multicast system for dynamic environments. In *3rd IEEE International Symposium on Network Computing and Applications (IEEE NCA)*, 2004.

[23] T.-W. J. Ngan, D. S. Wallach, and P. Druschel. Incentives-compatible peer-to-peer multicast. In *2nd Workshop on the Economics of Peer-to-Peer Systems*, 2004.

[24] P. Rodriguez, S.-M. Tan, and C. Gkantsidis. On the feasibility of commercial, legal p2p content distribution. In *ACM/SIGCOMM CCR*, 2006.

[25] R. Sherwood, R. Braud, and B. Bhattacharjee. Slurpie: A cooperative bulk data transfer protocol. In *Proceedings of IEEE INFOCOM*, 2004.

[26] J. Shneidman, D. Parkes, and L. Massoulie. Faithfulness in internet algorithms. In *Proceedings of SIGCOMM Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems (PINS'04), Portland, OR, USA*, 2004.

[27] M. Sirivianos, X. Yang, and S. Jarecki. Dandelion: Secure Cooperative Content Distribution with Robust Incentives. In *First Workshop on the Economics of Networked Systems (NetEcon)*, 2006.

[28] A. Steger and N. Wormald. Generating random regular graphs quickly. *Combinatorics, Probab. and Comput*, 8:377–396, 1999.

[29] N. Wormald. Models of random regular graphs. *Surveys in Combinatorics*, 276:239–298, 1999.