EquiCast: Scalable Multicast with Selfish Users

Idit Keidar Roie Melamed Ariel Orda

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.

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 [18, 5]. However, these networks suffer from the problem of "freeloaders", i.e., users who consume resources without contributing their fair share [2]. 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., [5, 10], typically rely on user altruism. For example, a node is expected to upload data blocks to other nodes for no return whenever it has available bandwidth [5, 10]. Hence, current incentive-based P2P systems do not solve the problem of "freeloaders" [12], 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 [12, 10].

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 shall allow unlimited downloads and uploads, while lower tier pricing plans shall limit traffic bandwidth [18]. With such a pricing scheme, users will most likely cease to be altruistic [18]. This may lead to low P2P system availability [13, 12] or even system collapse [2, 17]. 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 free-loading 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 [9]. More specifically, the goal of each node is to receive all the multicast packets while minimizing its sending rate. We restrict the strategies a node can choose to *protocol-obedient* ones, where a protocol-obedient strategy determines how many connections the node maintains and how many packets it sends on each connection, though it does not include hacking the protocol's code or assuming that others do so. We believe that this set of strategies is reasonable, since users usually do not have the technical knowledge required in order to modify an application code. Our formal model and cost function are presented in Section 3, and in Section 5.1 we formally define the set of protocol-obedient strategies.

In EquiCast, a single distribution server *S* (which can be implemented by multiple machines acting as one logical server) organizes the nodes into a static *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, as opposed 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 [3].

In Section 5, we prove that, in environments in which all the nodes are selfish, EquiCast disseminates all the multicast packets to all the nodes. Additionally, every protocol-obedient strategy in which a node exclusively cooperates with all its neighbors *strictly dominates* every protocol-obedient strategy in which it does not. 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

We are familiar with only two previous P2P multicast protocols for selfish environments [17, 11]. Ngan et al. [17] propose an incentive-based multicast protocol 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. Habib and Chuang [11] propose an incentive-based protocol for media streaming, in which 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.

Several previous distributed Internet services such as content distribution [5, 10], storage [6], and lookup [4] reward cooperation to some extent by incentivizing cooperative behavior. The BitTorrent [5] and Avalanche [10] 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 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 SAMARA storage system [6], each node is required to contribute as much disk space to the system as it is using, and in the GIA lookup system [4] 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 [2], 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 [2], since the reputation system continuously communicates with all the nodes.

Finally, *cost-sharing* multicast solutions e.g., [7], 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 $10{,}000$. *S* knows all the node identities, e.g., by each node registering itself at *S*.

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 at most $p+kc$ packets per δ time units, where k and c are small constants such that $k\geq 3$, $c\geq 4$, and $p\%k=0$. In addition, we require that $(k^2-k)(c-3)<p$ and $k^2(c-2)-2k<p$, in order to prove that every protocol-obedient strategy in which a node exclusively cooperates with all its neighbors strictly dominates every protocol-obedient strategy 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.

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* out of a set of possible *protocol-obedient strategies*. We defer the definition of this set to Section 5.1, since it relies on the protocol's code described in Section 4.3. Generally speaking, a protocol-obedient strategy must run the protocol as is and can only determine how many connections to maintain and how many packets to send.

Each node is selfish and *rational*, i.e., n_i chooses a strategy st_i that minimizes its selfish cost as defined below. A strategy A *strictly dominates* another strategy B if choosing A always incurs a lower cost than choosing B, regardless of the strategies chosen by other nodes. A *strongly dominating 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.

Problem statement. Our goal is to design a scalable P2P multicast protocol, in which if each node chooses a dominating strategy out of the set of protocol-obedient ones, then each node receives all the multicast packets. A second goal is efficiency, i.e., the maximal and expected sending/recieving overhead incurred on each node is $P(1+\frac{c\cdot k}{p})$ and $P(1+\frac{\epsilon}{p})$ packets, respectively, where ϵ does not depend on p.

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. For $k \geq 3$, a *k-regular random graph*[1] satisfies these properties with high probability [20, 8, 14]. *S* constructs the overlay, e.g., using one of the constructions in [20], and sends to each node the identities of its overlay neighbors, henceforth, simply called *neighbors*. Note 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 dominating 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 for each node n, *S* sends copies of these p packets to n with a probability of $\frac{k}{N}$, so that, on average, each data packet is sent to k 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 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 [16], 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

[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.

$\hat{n}$'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. Note that a node cannot optimize its balance 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.

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*.

On the one hand, the allowed imbalance should be large enough to reduce the probability of a cooperative node reaching L, in order to avoid over-loading *S*. 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 the value of L is independent of all the other system parameters.

4.3 Detailed description

4.3.1 The source protocol

Each round, *S* creates p new data packets, and for each node n it sends a copy of these packets to n with a probability of $\frac{k}{N}$. In the rare case in which, at a given round, no node is chosen to receive copies of the p new data packets, *S* restarts the round. Note that this does not add to the round duration, since computation time is zero.

Upon receiving a request from a node n to send x data packets to another node $\hat{n}$, *S* verifies that: (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 $\frac{k}{N}$.[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 at the end of the multicast session, we use the randomized back-off strategy described in [19].

4.3.2 The node protocol

Fig. 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.

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.

The pseudo-code of the node's protocol is presented is Fig. 2. It consists of four phases, which are executed sequentially.

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 1–5).

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 terminates its connection with that neighbor (lines 6–8). 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 terminates its connection with that neighbor (lines 19–21). Then, the node sends data packets to each of its neighbors. It sends as many requested packets as possible, 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 21). 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 at 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 23–25).

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 28–32). 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, at the beginning of

[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 $t+1$.

```
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 ⟨GOSSIP,$p$⟩ to $n$
5.     wait $\Delta$ time

Phase II (process gossip, send requests)
6. foreach $n \in neighbors$ from which
       no GOSSIP packet arrived
7.         disconnect from $n$
8.         contact S and ask for an alternative neighbor
9. $ids \leftarrow$ set of identifiers received in gossip packets, whose
           corresponding data packets were not received yet
10. foreach $n \in neighbors$ $reqs[n] \leftarrow \emptyset$
11. foreach $id \in ids$
12.     $id\_set \leftarrow$ set of neighbors that gossiped about $id$
13.     $ne \leftarrow$ a random neighbor from $id\_set$
            so that $|reqs[ne]| < \frac{p}{k} + c - 3$
14.     if there is no such $ne$ then continue
15.     $reqs[ne] \leftarrow reqs[ne] \cup \{id\}$
16. foreach $n \in neighbors$
17.     send ⟨REQUEST,$reqs[n]$⟩ to $n$
18. wait $\Delta$ time

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

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

Figure 2: Code for EquiCast node.

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, then 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 that $|L|k$ data packets in the future is 1. Therefore, for every constant $const$, $\frac{const}{R}$ is negligible, and for simplicity is assumed to be 0.

In Section 5.1 we define the set of *protocol-obedient strategies*, and in Section 5.2 we prove that every protocol-obedient strategy in which a node cooperates with all its neighbors strictly dominates every protocol-obedient strategy in which it does not. Additionally, we prove that if a node chooses such a dominating strategy, then it receives all the multicast packets.

5.1 The set of protocol-obedient strategies

We say that a connection is *maintained* between two neighboring nodes n and $\hat{n}$, if both n's and $\hat{n}$'s strategy is to be connected to each other assuming the other node does so too. That is, both n and $\hat{n}$ send the necessary packets in order to avoid a situation where the protocol dictates that the connection be terminated. Note that n and $\hat{n}$ can be either real nodes or nodes emulated by *S*.

Definition 1 (Protocol-obedient strategy). *A node's strategy is* protocol-obedient *if (i) the node runs the protocol's code described in Fig. 2 without changing any of the protocol's parameters except H; and (ii) the node maintains connections only with nodes whose identities are received from S.*

We believe that this set of strategies is reasonable for the common user, since such a user usually does not have the technical knowledge to modify an application code. Moreover, 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 [5, 12]. Note that we do allow a node to decide whether or not to send each packet. This allows a node to disconnect from a neighbor by simply not sending the necessary packets. Since such behavior is not supported by the code of most P2P applications, our assumptions about protocol-obedient strategies are less restrictive than theses applications' codes.

5.2 Proof of cooperation

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 $X_t(n,\hat{n}) = \sum_{i=0}^{i=t} x_t(n,\hat{n})$.

The following three lemmas are technical, and their proof is deferred to Appendix A.

Lemma 1. *At the end of each round, for every two neighboring nodes n and $\hat{n}$, $n.my\_balance[\hat{n}] = \hat{n}.neighbor\_balance[n]$.*

Lemma 2. *Assume that a node n is connected to another node $\hat{n}$. If both n's and $\hat{n}$'s strategy is maintaining the connection between them, then this connection is maintained.*

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

Lemma 4. *If a node n maintains connections with k nodes throughout the multicast session, then it receives from its neighbors and from S on behalf of its neighbors at least $P+Lk$ data packets (recall that L is negative).*

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 between $\frac{p}{k}+2$ and $\frac{p}{k}+c$ packets.*

Proof. The overhead of maintaining a connection consists of: (i) data overhead (X_R), i.e., packets that contribute to the node's balance with respect to the neighbor, (ii) gossip/request packets, and (iii) penalty packets.

According to the protocol (Fig. 2, lines 21–25), the maximum data overhead is $\frac{p}{k}+c-3$ data packets per-round. 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, 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, 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 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 maintains connections with more than $k{-}1$ nodes for 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 throughout the multicast session, then $f_n{<}\infty$ and $\frac{f_n}{R}{\leq}p{+}kc$.*

Proof. We first note that n can always maintain connections with k nodes if it chooses to do so, since n can always maintain connections with k nodes emulated by *S*. Hence, according to Lemma 4, if n maintains connections with k nodes throughout the multicast session, then it 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 at the end of the multicast session does not increase the per-round overhead. Thus, $\frac{f_n}{R}{\leq}p{+}kc$. □

We now discuss a scenario in which a node n maintains connections with more than k nodes throughout the multicast session. Note that n can maintain up to $2k$ connections if it maintains connections with all of its initial neighbors and with k emulated nodes as well. However, 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 additional nodes other than its neighbors can only increase n's cost.

Lemma 8. *Every protocol-obedient strategy in which a node n maintains connections with k nodes throughout the multicast session strictly dominates every protocol-obedient 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 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, $p{+}kc{<}(k{+}1)(\frac{p}{k}{+}2)$. Therefore, maintaining k connections throughout the multicast session incurs a lower cost than the cost of maintaining $k{+}1$ or more connections throughout the multicast session. □

The following lemma shows that it is preferable for a node to maintain connections with k neighbors throughout the multicast session.

Lemma 9. *Every protocol-obedient strategy in which a node n maintains connections with k nodes throughout the multicast session strictly dominates every protocol-obedient strategy in which n maintains connections with j nodes, where $j{\neq}k$.*

Proof. We 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 theorem follows from Lemmas 6, 7, and 8. □

We next show that a node benefits more from connections with its original k neighbors than from emulated ones.

Lemma 10. *Assume that a node n maintains a connection with a non-emulated node $\hat{n}$. 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, 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 $\hat{n}$. Hence, 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 dominating strategies out of the set of protocol-obedient strategies, 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 maintaining 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. □

We have shown that it is beneficial for nodes to maintain their connections with their neighbors. We now show that it is also beneficial for them to *cooperate* with their neighbors, by maintaining a non-negative balance. That is, each node sets its H parameter to be equal to or larger than 0. This reduces the probability for nodes reaching a balance lower than L, and hence limits the overhead on S.

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, at 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 $\hat{n}$ at 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. Below, we detail only the differences between the two versions.

Architecture. DEC is deployed on top of a dynamic overlay that supports node joins and leaves. For example, we can use the overlay in [15], which is a dynamically maintained k-regular graph composed of $\frac{k}{2}$ Hamiltonian cycles.

The cost function. DEC's cost function is obtained from EquiCast's 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.

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 [15]. 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 at 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, at 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, at 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 $m{\cdot}p$ data packets, where m is the number of rounds during which n_1 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 [15]. For example, assume that, prior to n's leave, n was connected to nodes n_1 and n_2, and n_1 and n_2 become neighbors after n's leave. 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 at 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 leave. Therefore, at 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, at 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 at the end of round t as B_{n1} and B_{n2}, respectively. If $B_{2n}{>}B_{n1}$, then n_1 may not receive $m{\cdot}p$ data packets, where m is the number of rounds during which n_1 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.

Finally, a node n' that is connected to the overlay for m rounds may receive less than $m{\cdot}p$ data packets if it has negative incoming balances with respect to its neighbors on leave time. Hence, after it leaves the overlay, n' can receive up to $|L|k$ data packets from *S* in return for sending *S* a fine packet for each data packet.

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 dominating strategy. We are unaware of any previous P2P multicast protocol that is proven 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. Blanc, Y.-K. Liu, and A. Vahdat. Designing incentives for peer-to-peer routing. In *Proceedings of the IEEE Infocom Conference*, 2005.

[3] 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.

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

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

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

[7] 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.

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

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

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

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

[12] 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.

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

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

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

[16] 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.

[17] 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.

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

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

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

A Proof of Cooperation: Basic Lemmas

Lemma 1. *At the end of each round, for every two neighboring nodes n and $\hat{n}$, $n.my\_balance[\hat{n}] = \hat{n}.neighbor\_balance[n]$.*

Proof. By induction.
Base: $t = 0$. $n.my\_balance[\hat{n}] = \hat{n}.neighbor\_balance[n] = 0$.
Step: Assume that, at the end of round t, $n.my\_balance[\hat{n}] = \hat{n}.neighbor\_balance[n]$. We will prove that, at 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 Fig. 2, lines 22 and 32). 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 Fig. 2, lines 22, 26, and 32).

When n sends d new data packets to $\hat{n}$, $n.my\_balance[\hat{n}]$ is increased by d (see Fig. 2, line 22). 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 $t{+}1$, as $\hat{n}$ ignores unsolicited data packets (see Fig. 2, line 29). Therefore, upon receiving the d data packets, $\hat{n}$ increases $\hat{n}.neighbor\_balance[n]$ by d.

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 Fig. 2, lines 23–25). Additionally, $n.my\_balance[\hat{n}]$ is increased by w (see Fig. 2, line 26). 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 the protocol, $\hat{n}.neighbor\_balance[n]{<}L{+}\frac{p}{k}$ at 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 Fig. 2, lines 30–32). 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 Fig. 2, line 30), 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. *Assume that a node n is connected to another node $\hat{n}$. If both n's and $\hat{n}$'s strategy is maintaining the connection between them, then this connection is maintained.*

Proof. Without loss of generality, we will prove that $\hat{n}$ does not terminate the connection with n. Since $\hat{n}$'s strategy is maintaining the connection with n, then it terminates the connection with n in a given round t if: (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) n did not send to $\hat{n}$ a fine packet in round t and either $\hat{n}$ is an emulated node or, at the end of round t, $\hat{n}.neighbor\_balance[n]{<}0$; or if (iv) at the end of round t, $\hat{n}.neighbor\_balance[n]{<}L$.

Since n's strategy is maintaining the connection with n, then (i), (ii), and (iii) do not happen. In addition, we note that n can ensure that, at 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 $X_{t+1}(n,\hat{n}) = \frac{(t+1)p}{k} + b_{t+1}(n,\hat{n})$.

$X_{t+1}(n,\hat{n}) = X_t(n,\hat{n}) + x_{t+1}(n,\hat{n}) = \frac{tp}{k} + b_t(n,\hat{n}) + x_{t+1}(n,\hat{n})$. From the code (Fig. 2, lines 22 and 26), we know that: $b_{t+1}(n,\hat{n}) = b_t(n,\hat{n}) + x_{t+1}(n,\hat{n}) - \frac{p}{k}$. Therefore, $b_t(n,\hat{n}) + x_{t+1}(n,\hat{n}) = b_{t+1}(n,\hat{n}) + \frac{p}{k}$. Hence, $X_{t+1}(n,\hat{n}) = \frac{(t+1)p}{k} + b_{t+1}(n,\hat{n})$. □