Be Fair and Be Selfish! Characterizing Deterministic Diffusive Load-Balancing Schemes with Small Discrepancy

Petra Berenbrink, Ralf Klasing, Adrian Kosowski, Frederik Mallmann-Trenn, Przemyslaw Uznanski

Be Fair and Be Selfish!

Characterizing Deterministic Diffusive Load-Balancing Schemes with Small Discrepancy

Petra Berenbrink[1], Ralf Klasing[2], Adrian Kosowski[3], Frederik Mallmann-Trenn[1], and Przemysław Uznański[4]

[1]Simon Fraser University, {petra,fmallman}@sfu.ca
[2]CNRS – LaBRI – Université de Bordeaux, ralf.klasing@labri.fr
[3]Inria – LIAFA – Paris Diderot University, adrian.kosowski@inria.fr
[4]LIF – Aix Marseille University, przemyslaw.uznanski@lif.univ-mrs.fr

Abstract

We consider the problem of deterministic distributed load balancing of indivisible tasks in the discrete setting. A set of n processors is connected into a d-regular symmetric network. In every time step, each processor exchanges some of the tasks allocated to it with each of their neighbours in the network. The goal is to minimize the discrepancy between the number of tasks on the most-loaded and the least-loaded processor as quickly as possible. In this model, the performance of load-balancing schemes obtained by rounding the continuous diffusion process up or down to the nearest integer was considered by Rabani et al. (1998), who showed that after $T = O(\log(Kn)/\mu)$ steps any such scheme achieves a discrepancy of $O(d \log n/\mu)$, where μ is the spectral gap of the transition matrix of the network graph, and K is the initial load discrepancy in the system.

In this work, we identify natural additional conditions on the form of the discrete deterministic balancing scheme, which result in smaller values of discrepancy between maximum and minimum load. Specifically, we introduce the notion of a *cumulatively fair* load-balancing scheme, in which every node sends in total almost the same number of tasks along each of its outgoing edges during every interval of consecutive time steps, and not only at every single step. As our first main result, we prove that any scheme which is cumulatively fair and which has the selfish property that every node retains a sufficient part of its load for itself at each step, achieves a discrepancy of $O(\min\{d\sqrt{\log n/\mu}, d\sqrt{n}\})$ in time $O(T)$, and that in general neither of these assumptions may be omitted without increasing discrepancy. We then show that any cumulatively fair scheme satisfying additional assumptions on fairness and selfishness achieves a discrepancy of $O(d)$ almost as quickly as the continuous diffusion process. In particular, we provide an example of a scheme belonging to this class, which is both deterministic and stateless, i.e. such that the number of tasks sent by a node to its neighbours is a function of only its current load. The $O(d)$ discrepancy achieved by this scheme is asymptotically the best possible in the class of stateless schemes and is reached after time $O(T + \log^2 n/\mu)$. We also provide time bounds for other schemes belonging to the characterized class, such as variants of the deterministic rotor-router process, and remark on possible generalizations of our results.

1 Introduction

Load balancing is an important aspect of efficient use of massively parallel computations. A constraint on balancing schemes for large distributed networks is the requirement of *locality*, meaning that the nodes of the network should be able to make their balancing decisions by knowing only the local load situation. Additionally, it is advantageous that nodes balance their workload with nearby nodes only, instead of sending their load to, say, randomly chosen nodes somewhere in the network.

In this paper, we analyze so-called *diffusion-based neighbourhood load balancing algorithms*, where nodes (processors) balance their load only with their direct neighbours. Our balancing algorithms do not rely on any global knowledge, and they do not even need to know the load of their direct neighbours. We assume that the processors are connected by an arbitrary d-regular graph G. At the beginning, every node has a certain number of tokens, representing its initial load. The goal is to distribute the tokens in the network graph as evenly as possible. More precisely, we aim at minimizing the *discrepancy* (sometimes also referred to in the literature as *smoothness*), which is defined as the difference between the maximum load and the minimum load, taken over all nodes of the network.

In general, diffusion-based neighbourhood load balancing algorithms operate in parallel, in synchronous steps. In each step, every node balances its load with all of its neighbours. One distinguishes between *continuous* load balancing models, in which load can be split arbitrarily, and the much more realistic *discrete* model, in which load is modeled by tokens which cannot be split. In the former case, the standard continuous diffusion algorithm works as follows. Every node u having load $x(u)$ considers all of its neighbours at the same time, and sends $x(u)/(d+1)$ load to each of its d neighbours, keeping $x(u)/(d+1)$ load for itself. This process may also be more efficiently implemented as follows: for each neighbour v of u, node u calculates the load difference $x(u)-x(v)$ between them and divides the outcome by $d+1$. If $x(u)-x(v)>0$, then it sends exactly $(x(u)-x(v))/(d+1)$ load to v. It is well-known (cf. e.g. [16]) that the load in the continuous model will eventually be perfectly balanced. In the discrete case with indivisible tokens, exact simulation of the continuous process is not possible. A node u may instead try to round the value $(x(u)-x(v))/(d+1)$ up or down to an integer. Discrete balancing approaches are, in general, much harder to analyze than continuous algorithms. One of the main analysis techniques, a new variant of which we also apply in this paper, is to compare the discrete algorithm to the continuous version and to bound the so-called error that occurs due to the rounding.

In [14], Rabani et al. showed that for a wide class of discrete balancing algorithms, the discrepancy is bounded by $O(d\log n/\mu)$ after $T=O(\log(Kn)/\mu)$ steps, where μ is the eigenvalue gap of the transition matrix of the underlying Markov Chain. This result holds for all regular graphs and can, in fact, be adapted to non-regular graphs. It applies to any discrete load balancing scheme which, at every time step, rounds the load which would be exchanged in the continuous diffusion process by a given pair of nodes to one of the nearest integers, either up or down. The time T is also the time in which a continuous algorithm balances the system load (more or less) completely. T can also be regarded as the *mixing time* of a random walk on G, which is the time it takes for a random walk to be on every node with the same probability. Within the class of schemes considered in [14], the bound of $O(d\log n/\mu)$ on discrepancy cannot be improved for many important graph classes, such as constant-degree expanders. Since the work [14], many different refinements and variants of this approach have been proposed [4,8,9,15], as well as extensions to other models, including systems with non-uniform tasks [4] and non-uniform machines [2].

1.1 Our Contribution

The main contribution of our paper consists in putting forward conditions on discrete load-balancing schemes, which lead to significantly improved bounds on discrepancy with respect to the (less restrictive) conditions from [14]. Our schemes include numerous algorithms, and the only requirements are fairness conditions which restrict the difference between the number of tokens that a node can send out over different edges, and a selfishness requirement saying that all nodes have to keep a certain fraction of their load for themselves. Among schemes satisfying our conditions, we focus on two important families of load-balancing algorithms. The first family is that of stateless algorithms, denoted $\textsc{Send}(f(x))$, in which the number of tokens sent out by a node to each of its neighbours is given by the same deterministic function $f:\mathbb{N}\to\mathbb{N}$ of its current load x. The second family relies on the principle of round-robin distribution of tokens to neighbours in variants

of the so-called ROTOR-ROUTER model (also referred to in the literature as the Propp model, [3, 6, 10]).

The first of the conditions we propose describes algorithms which are *cumulatively fair*, i.e., such that, informally speaking, for every interval of consecutive time steps, the total number of tokens sent out by a node differs by at most a small constant for any two adjacent edges. We show that algorithm satisfying the cumulative fairness condition achieve $O(d \cdot \min\{\sqrt{\log n/\mu}, \sqrt{n}\})$-discrepancy in $O(T)$ time steps, if at least half of the edges in the graph are self-loop edges. This provides an improvement with respect to the general result from [14]: for example, for expanders, the achieved discrepancy after time $O(T)$ is $O(\sqrt{\log n})$, as opposed to $\Theta(\log n)$. Our techniques for the analysis of cumulatively fair algorithms rely on a comparison between the behaviour of the cumulative fair algorithm, and that of a Markovian process governed by the transition matrix of the graph. This comparison is, however, applied to long time intervals, as opposed to single time steps, and additionally takes advantage of properties of random walks in graphs.

By defining additional constraints on cumulatively fair algorithms, we obtain even tighter bounds on the discrepancy. Specifically, we will call a cumulatively fair algorithm *strictly fair* if, in every step, the number of tokens that are sent out over every edge incident to a node differs by at most one, and we will call it *selfish* if for each node, at least one self-loop edge adjacent to it may not receive fewer tokens than any adjacent non-self-loop edge. For strictly fair and selfish algorithms, we show that an $O(d)$-discrepancy is achieved within $O(T + d \log^2 n/\mu)$ time steps. This time, our techniques combine the algebraic techniques previously developed for cumulatively fair algorithms with a more combinatorial view of the system, including an analysis of potential functions defined on nodes. Algorithms which meet our conditions of strict fairness and selfishness include some variants of the rotor-router approach (which we subsequently denote by the symbol ROTOR-ROUTER$*$), as well as some algorithms from the stateless SEND family. One particularly representative example of such an SEND-type algorithm is the SEND$\left(\left\lceil \frac{x}{3d} - \frac{2}{3} \right\rceil\right)$ scheme, in which a node with current load x sends exactly $\left\lceil \frac{x}{3d} - \frac{2}{3} \right\rceil$ to each of its d neighbours, and keeps all the remaining tokens for itself. For this algorithm, we show that it can be seen as strictly fair process with a large number of selfish self-loops, and as a result it achieves $O(d)$-discrepancy d times faster than for the general selfish class, namely in $O(T + \log^2 n/\mu)$ steps.

The overall message of our paper is that by imposing the right fairness and selfishness constraints, we can obtain very simple and practical deterministic load-balancing algorithms which can match or surpass all, frequently more complex, results in the state-of-the-art of the literature. The discrepancy achieved by our best algorithms, which is $O(d)$, appears to be the best possible gap achievable in reasonably short time in many graph classes (e.g., in polylogarithmic time for expanders). In fact, we provide matching lower bounds of $\Omega(d)$ on the eventual discrepancy of some of the most natural (ROTOR-ROUTER-type and SEND-type) algorithms which we consider. For a detailed exposition of our results, we refer the reader to Section 2 and Table 1.

The remainder of this paper is structured in the following way. In the rest of the introduction, we set our results in the context of the state-of-the-art of the literature. In Section 2, we provide a formalization of the model, which allows us to state the required properties of cumulative fairness, strict fairness, and selfishness. We then formulate our main theorems for algorithms satisfying these properties, and provide examples of algorithms satisfying them. Sections 3 and 4 contain proofs of our main results on the discrepancy of cumulatively fair algorithms and of strictly fair selfish algorithms, respectively. Lower bounds on the discrepancy of some of the studied processes are presented in Section 5. We conclude with some comments on possible extensions of our results in Section 6.

1.2 Comparison with the State-of-the-Art

Within the framework we consider, we provide the first systematic improvement with respect to the rounding strategy used in [14], achieving deterministic discrete load-balancing schemes with smaller discrepancy and in the same time $O(T)$. For the specific families of ROTOR-ROUTER-type and SEND-type algorithms, which have been considered previously in the load-balancing literature (in [11, 17] and [2], respectively), we also provide a significant improvement with respect to the state-of-the-art regarding discrepancy and/or the time required to achieve it.

When comparing performance characteristics of load-balancing schemes (i.e., their eventual discrepancy bound and the time required to reach it), our approach is superior to most prior work. A notable exception is the algorithm of [15], which can, in particular, be deployed in constant-degree graphs ($d = O(1)$) in the

Algorithm	Discrepancy after time $O(T)$	Discrepancy later in time: Discrepancy	Discrepancy later in time: Time required	Ref.	D	SL	NL	NC
Discrete diffusion with arbitrary rounding on edges	$O\left(\frac{d\log n}{\mu}\right)$			[14]	✓	✓	✓	✓
Randomized distribution of extra tokens by vertices	$O\left(d\sqrt{\log n}+\sqrt{\frac{d\log n\log d}{\mu}}\right)$			[5]	X	✓	✓	✓
	$O\left(d^2\sqrt{\log n}\right)$			[15]				
Randomized rounding to nearest integers on edges	$O\left(\sqrt{d\log n}\right)$			[15]	X	✓	X	✓
Cumulatively fair algorithms	$O\left(d\min\left\{\sqrt{\frac{\log n}{\mu}},\sqrt{n}\right\}\right)$			Thm 2.5				
• Rotor-Router	"			Thm 2.5	✓	✓	✓	✓
• Send$\left(\left\lfloor\frac{x_t(u)}{cd}\right\rfloor\right)$, for any $c\geq 2$:	"			Thm 2.5	✓	✓	✓	✓
for $c=4$:	"	$O(d\,\mathrm{diam})$	$O\left(\frac{d^3n}{\mu}\right)$	Thm 2.5, [2]	✓	✓	✓	✓
• **Strictly fair and selfish algorithms**	"	$O(d)$	$O\left(T+\frac{d\log^2 n}{\mu}\right)$	Thm 2.6				
• Rotor-Router*	"	$\Theta(d)$	"	Thm 2.6	✓	✓	✓	✓
• Send$\left(\left\lceil\frac{x_t(u)}{2d}-\frac{1}{2}\right\rceil\right)$	"	$\Theta(d)$	"	Thm 2.6	✓	✓	✓	✓
• Send$\left(\left\lceil\frac{x_t(u)}{3d}-\frac{2}{3}\right\rceil\right)$	"	$\Theta(d)$	$O\left(T+\frac{\log^2 n}{\mu}\right)$	Thm 2.6	✓	✓	✓	✓
• Computation based on continuous diffusion	$\Theta(d)$			[4]	✓	X	X	X

Legend: D — Deterministic process; SL — Stateless process; NL — Cannot produce negative loads; NC — No additional communication required.

n — number of nodes of the graph, T — load balancing time of the continuous diffusion process, μ — spectral gap of graph transition matrix, diam — graph diameter.

Table 1: A comparison of the discrepancy of load-balancing algorithms in the diffusive model for d-regular graphs. In all result statements, we assume that the graph is augmented with at least d self-loops per node.

so-called dimension exchange model to achieve a discrepancy of $O(1)$ in $O(T)$ steps *with high probability*, whereas our best algorithms may, in the pessimistic case, require up to $O(T\log n)$ steps to reach such a gap *deterministically* in the diffusive model. The authors of [15] also present randomized algorithms for the diffusive model. They achieve after $O(T)$ time a discrepancy of $O(d^2\sqrt{\log n})$: They first send $\lfloor x(u)/(d+1)\rfloor$ tokens to every neighbour and itself, and afterwards they distribute the remaining tokens randomly. Additionally, they provide an algorithm which achieves after $O(T)$ time a discrepancy of $O(\sqrt{d\log n})$ by rounding the flow sent over edges randomly to the nearest integers which might cause negative loads (see Section 1.3 for a detailed discussion).

Finally, we remark that the only other result in the literature known to the authors, which achieves a discrepancy of $O(d)$ in short time ($O(T)$ steps) in the diffusive model, is presented in [4]; however, their algorithm relies on extensive communication, computation and memory capabilities of nodes over multiple time steps, as well as certain additional assumptions on minimum load of nodes, and as such does not really belong to the same category as our algorithms (cf. Table 1 for a comparison of properties).

In our work, we have presented our analysis of load balancing schemes for arbitrary d-regular graphs, in the diffusive model, and with unit-size tasks. All of our considerations concern the static scenario, in which the network does not change and no new load is added over time. Nevertheless, the Rotor-Router-type and Send-type algorithms which we study are resilient to changes in the network topology, and treat any current state of the network as a valid starting state for the load-balancing process.

1.3 Related Work

Herein we only consider related results for load balancing in the discrete diffusive and balancing circuit models, and some results for the rotor-router model which are relevant to our work.

Diffusive load balancing. Discrete load balancing processes have been studied in numerous works since [14]. The authors of [8] propose a deterministic load balancing process in which the continuous load transfer on each edge is rounded up or down deterministically, such that the sum of the rounding errors on each edge up to an arbitrary step t is bounded by a constant. This property is called the *bounded-error property*. Then they show that after T steps any process with bounded-error property achieves a discrepancy of $O(\log^{3/2} n)$ for hypercubes and $O(1)$ for constant-degree tori. There are no similar results for other graph classes. Note

that the algorithm of [8] has the problem that the outgoing demand of a node might exceed its available load, leading to so-called *negative load*.

In [3], the authors consider ROTOR-ROUTER-type walks as a model for load balancing. It is assumed that half of the edges of every node are self-loops. The authors present an algorithm which falls in the class of *bounded-error diffusion processes* introduced in [8]. This results in discrepancy bounds of $O(\log^{3/2} n)$ and $O(1)$ for hypercube and r-dimensional torus with $r = O(1)$. In [2], the authors consider the diffusion algorithms that always rounds down for heterogenous networks. They also show that a better load balance can be obtained when the algorithm is allowed to run longer than T steps.

In [4], the authors propose an algorithm that achieves discrepancy of $2d$ after T steps for any graph. For every edge e and step t, their algorithm calculates the number of tokens that should be sent over e in t such that the total number of tokens forwarded over e (over the first t steps) stays as close as possible to the amount of load that is sent by the continuous algorithm over e during the first t steps. However, their algorithm can result in negative load when the initial load of any node is not sufficiently large and it has to calculate the number of tokens that the continuous algorithm sends over all edges. Note that the algorithm presented in this paper has to simulate the continuous algorithm in order to calculate the load that it has to transfer over any edge, whereas our algorithms are much easier and they do not need any additional information, not even the load of their neighbours.

There are several publications that suggest randomized rounding schemes [1, 3–5, 8, 15] to convert the continuous load that is transferred over an edge into discrete load. The randomized algorithm of [8] achieves an imbalance bound of $O(d \log\log n/\mu)$ for d-regular graphs, $O(d \log\log n)$ for expanders, $O(\log^2 n \log\log n)$ for hypercubes, and $O(n^{1/2d} \log\log n)$ for tori. The algorithm of [5] calculates the number of *additional* tokens (the difference between the continuous flow forwarded over edges and the number of tokens forwarded by the discrete algorithm after rounding *down*). All additional tokens are sent to randomly chosen neighbours. Their discrepancy bounds after T steps are $O(d \log\log n/\mu)$ and $O(d\sqrt{\log n} + \sqrt{d \log n \log d/\mu})$ for d-regular graphs, $O(d \log\log n)$ for expanders, $O(\log n)$ for hypercubes, and $O(\sqrt{\log n})$ for tori. In [4], the authors also consider a randomized version of their protocol for arbitrary graphs. The discrepancy bound is w.h.p. $O(\sqrt{d \log n})$ after T time steps if the initial load on every node is at least $d/4 + c\sqrt{d \log n}$ for some constant $c > 0$. The authors of [15] achieve a discrepancy of $O(d^2\sqrt{\log n})$. They also provide an algorithm with a discrepancy of $O(\sqrt{d \log n})$ which rounds the flow sent over edges randomly. Their algorithm can generate negative loads which can be avoided by adding artificial tokens at the beginning and removing them at the end. By doing so, the gap between the average load and the maximum load is $O(\sqrt{d \log n})$, but the discrepancy can be arbitrarily high since the minimum load can be 0.

Dimension exchange model. In the Dimension Exchange model, the nodes are only allowed to balance with one neighbour at a time. Whereas for all diffusion algorithms considered so far the discrepancy in the diffusion model is at least d, dimension exchange algorithms are able to balance the load up to an additive constant. In [9] the authors consider a discrete dimension exchange algorithm for the matching model. Every node i that is connected to a matching edge calculates the load difference over that edge. If that value is positive, the algorithm rounds it up or down, each with a probability one half. This result is improved in [15], where the authors show that a constant final discrepancy can be achieved within $O(T)$ steps for regular graphs in the random matching model, and constant-degree regular graphs in the periodic matching (balancing circuit) model.

Rotor-router walks. Originally introduced in [13], the *rotor-router walk* model was employed by Jim Propp for derandomizing the random walk, thereby frequently appearing under the alternative names of *Propp machines* and *deterministic random walks* [6,7,10,11]. In the rotor-router model, the nodes send their tokens out in a round-robin fashion. It is assumed that that the edges of the nodes are cyclically ordered, and that every node is equipped with a rotor which points to one of its edges. Every node first sends one token over the edge pointed to by the rotor. The rotor is moved to the next edge which will be used by the next token, and so on, until all tokens of the node have been sent out over one of the edges. In the next step, the token distribution starts at the position where the rotor pointed at the end of the previous step. It has been shown that the rotor walks capture the average behaviour of random walks in a variety of respects such as hitting probabilities and hitting times.

The rotor-router model can be used for load balancing, and directly fits into the framework we consider in this paper. In [3], the authors study a lazy version of the rotor-router process (half of the edges are self-loops) for load balancing. They prove that the rotor walk falls in the class of *bounded-error diffusion processes* introduced in [8]. Using this fact they obtain discrepancy bounds of $O(\log^{3/2} n)$ and $O(1)$ for the hypercube and r-dimensional torus with $r = O(1)$, respectively, which improve the best existing bounds of $O(\log^2 n)$ and $O(n^{1/r})$.

2 Model and Overview of Results

2.1 Model, Definitions, and Notation

The input graph. The input of the load-balancing process is a symmetric and directed regular graph $G = (V, E)$ with n nodes. Every node has out-degree and in-degree d. We have $m \in \mathbb{N}$ indivisible tasks (workload) which are arbitrarily distributed over the nodes of the network. For simplicity of notation only, we assume that initially G does not contain multiple edges. In general, the nodes of the graph may be treated as anonymous, and no node identifiers will be required. In some algorithms, we will assume that each node u maintains a local ordering of its neighbours within its neighbourhood, which is persistent over time. For the sake of analysis only, we number the nodes with consecutive integers from 1 to n, and identify nodes with their labels.

Time is divided into synchronized steps, numbered with integers starting from 1. Let $\mathbf{x}_t = (x_t(1), \ldots, x_t(n))$ be the *load vector* at the beginning of step t, where $x_t(u)$ corresponds to the load of node u at the beginning of step t. In particular, $\mathbf{x}_1$ denotes the initial load distribution and $\bar{\mathbf{x}}$ is the real-valued vector resulting when every node has achieved average load, $\bar{\mathbf{x}}(u) = \frac{1}{n}\sum_{u\in V} x_1(u) \equiv \bar{x}$, for all $u \in V$. Note that the total load summed over all nodes does not change in time. We will denote by K the maximal initial discrepancy in $\mathbf{x}_1$, i.e., $K = \max_{u\in V} x_1(u) - \min_{u\in V} x_1(u)$.

Methods of retaining load at a node. We introduce two different ways which enable nodes to keep a fraction of their own load, instead of distributing it to their neighbours: *self-loops* and so-called *remainders*. The difference between them manifests only in the analysis: we will be able to say that a node in a process keeps a part of its load in its remainder, and a part on its self-loops, in such a way that fairness conditions on outgoing edges and self-loops are preserved for this node.

Formally, in order to introduce self-loops, we transform G into the graph $G^+ = (V, E \cup E^\circ)$ by adding d° self-loops to every node. For a fixed node $u \in V$, let $E^\circ_u = \{e_1(u,u), \ldots, e_{d^\circ}(u,u)\}$ denote the set of self-loops of u and let E_u denote the outgoing edges of u in G. We assume $d^\circ = O(d)$. We can now define

$$E^+_u = E^\circ_u \cup E_u \quad \text{and} \quad E^\circ = \bigcup_{u\in V} E^\circ_u.$$

In the following, we call G the *original graph* and G^+ the *balancing graph*. We remark again that the balancing graph is introduced for purposes of analysis, only, and is completely transparent from the perspective of algorithm design. We also define $d^+ = d + d^\circ$ as the degree of any node in G^+. $N(u)$ is the set of direct neighbours of u in G^+, i.e., it contains all neighbours of u in G including u itself (because of the self-loops).

For a fixed edge $e = (u, v) \in E^+$, let $f_t(e) = f_t(u, v)$ be the number of tasks which u sends to v in step t. In particular, let $f_t(u, u) = \sum_{e\in E^\circ_u} f_t(e)$. Let $F_t(e)$ denote the cumulative load sent from u to v in steps $1, \ldots, t$, i.e, $F_t(e) = \sum_{\tau\le t} f_\tau(e)$. For a fixed node u, we define $f^{out}_t(u) = \sum_{v\in N(u)} f_t(u, v)$ to be the number of tasks (or flow) leaving u in step t. The incoming flow is then defined as $f^{in}_t(u) = \sum_{\{v:u\in N(v)\}} f_t(v, u)$. We define the cumulative incoming and outgoing flows $F^{out}_t(u)$ and $F^{in}_t(u)$ accordingly, and $\mathbf{F}^{out}_t$ and $\mathbf{F}^{in}_t$ are defined as the vectors of the (cumulative) outgoing and incoming flow. Note that for all $u \in V$ and all steps t

$$x_1(u) + F^{in}_{t-1}(u) = r_t(u) + F^{out}_t(u). \tag{1}$$

Moreover,

$$x_t(u) = f^{out}_t(u) + r_t(u). \tag{2}$$

The remainder $r_t(u)$ of node u in step t is the number of tokens of u that will not participate in the load distribution over its outgoing edges and self-loops. Then, $\mathbf{r}_t = (r_t(1), \ldots, r_t(n))$ denotes the *remainder vector* at step t, where $r_t(i)$ is the number of tokens kept by the i'th node of G in step t. We will denote by r the upper bound on the maximum remainder of an algorithm, satisfying $r_t(u) \leq r \leq d^+$ for every time step t and every $u \in V$. We say that an algorithm does not use any remainders if $r_t(v) = 0$ for all nodes v and steps t. We also say that u *keeps* $r_t(u) + \sum_{e \in E_u^\circ} f_t(e)$ tokens in step t. Some algorithms may make use of negative remainders ($r_t(v) < 0$), which are modeled by "negative tokens". We will only make use of such negative remainders, whenever they are compensated by the same number of artificially added positive tokens, retained on the self-loops of the same node, to compensate for the negative remainder. Thus, all such tokens appear in analysis only, and are transparent from an implementation perspective. For the analysis, we require the previously stated bound on $r_t(v)$ to hold also in the sense of absolute values, $|r_t(v)| \leq r$.

We say that an algorithm achieves a *c-discrepancy* if there exists a time step t such that for all $t' \geq t$ the load difference between the node with the *highest load* and the node with the *lowest load* at time t' is bounded by c. Formally, $\max_{u,v \in V}\{|x_{t'}(u) - x_{t'}(v)|\} \leq c$.

We say an algorithm is *c-unbalanced* if there exists a time step t such that for all $t' \geq t$ the gap between the node with the highest load and the *average load* is bounded by c. Formally, $\max_u\{x_{t'}(u) - \bar{x}\} \leq c$.

An algorithm A is called *stateless* if the load it sends over edges in any step depends solely on the load of the node at this time step.

2.2 Formulation of Fairness Properties

In this paper, we analyze a new and wide class of balancing algorithms motivated by rounding schemes. First we introduce two notions which restrict the difference in the number of tokens that a node can send out over its edges. The first notion implies that the amount of *cumulative* flow that is sent out over every edge of u (including the self-loops) up to step t can differ by at most a constant δ. Additionally, the number of tokens not being distributed over self-loops or outgoing edges has to be smaller than the degree of the balancing graph.

Definition 2.1. *An algorithm is called* cumulatively δ-fair *for some $\delta = O(1)$ if for all $t \in \mathbb{N}$, $u \in V$ we have*

- *all edges (including self-loops) $e_1, e_2 \in E_u^+$ satisfy $|F_t(e_1) - F_t(e_2)| \leq \delta$, and*
- *the remainder kept on all nodes is bounded by $|r_t(u)| \leq r \leq d^+$.*

The second notion is called *strict fairness*. Here we restrict the algorithm to distribute its load fairly among all edges such that the load difference of every pair of edges is at most 1 in every step.

Definition 2.2. *An algorithm is called* strictly δ-fair *for some $\delta = O(1)$ if for all $t \in \mathbb{N}$, $u \in V$ we have*

- *all outgoing edges $e_1, e_2 \in E_u$ satisfy $|F_t(e_1) - F_t(e_2)| \leq \delta$,*
- *all edges (including self-loops) receive $\lfloor x_t(u)/d^+ \rfloor$ or $\lceil x_t(u)/d^+ \rceil$ many tokens in step t, and*
- *no remainder is kept on nodes ($r_t(u) = r = 0$).*

We remark that in the definition of strictly fair algorithms, the cumulative fairness-type condition is only imposed on outgoing edges, and does not include self-loops. Nevertheless, we observe that strictly δ-fair algorithms are a subclass of cumulatively δ-fair algorithms. Every strictly δ-fair balancing algorithm admits an equivalent reformulation as a cumulatively δ-fair algorithm, which differs only in how load retained at a node is processed (moving it between self-loops and the remainder, which is indistinguishable on the original graph).

Proposition 2.3. *For any strictly δ-fair load-balancing algorithm A, there exists a cumulatively δ-fair load-balancing algorithm A', such that the actions of A and A' on the original graph G are indistinguishable. Moreover, the remainder used by A' fulfils $|r_t(u)| \leq r = d^\circ$.*

(For completeness, we provide a short proof of this Proposition in the Appendix.)

Finally, we call a strictly fair algorithm *selfish* if it favours self-loop edges over original edges. As we will see in Theorem 2.6, the "more selfish" an algorithm is, the faster it balances.

Definition 2.4. *Let $s \leq d^\circ$. An algorithm is called* s-selfish *if in every step t, at least s self-loops receive $\lceil x_t(u)/d^+ \rceil$.*

In the following subsections, we state our most important results for cumulatively fair and strictly fair selfish algorithms, and we give some specific examples of such processes.

2.3 Main Theorems of the Paper

We are now ready to formally state the main theorems of the paper. Recall that $T = O\left(\frac{\log K + \log n}{\mu}\right)$ is the balancing time of the continuous diffusion algorithm if the initial discrepancy is K, and d is the number of outgoing edges of each node.

Our first result concerns cumulatively fair algorithms. We show bounds on the discrepancy of such algorithms after time $O(T)$, which improve upon the discrepancy bound of $O(\frac{d \log n}{\mu})$ given by [14] for a slightly more general class of algorithms.

Theorem 2.5. *Let A be any cumulatively fair algorithm. For any d-regular input graph G, after $O\left(\frac{\log(Kn)}{\mu}\right)$ time, A achieves:*

(i) $O\left(d \cdot \sqrt{\frac{\log n}{\mu}}\right)$*-discrepancy for* $d^+ \geq 2d$.

(ii) $O\left(d \cdot \sqrt{n}\right)$*-discrepancy for* $d^+ \geq 2d$.

(iii) $O\left(d \cdot \frac{\log n}{\mu}\right)$*-discrepancy for arbitrary* $d^+ \geq d + 1$.

where d^+ is the degree of the balancing graph (including self-loops) used by A.

Claim (i) of the theorem shows that our algorithms achieve a better discrepancy after T steps compared to the result of [14] if the number of self-loops is at least as big as the number of original edges d. (For a smaller number of self-loops, we were unable to show the same discrepancy without increasing the required time.) Claim (ii) provides an improvement for graphs with a bad expansion (small eigenvalue gap), such as cycles. The proof of the theorem is deferred to Section 3.

We remark that when providing improved bounds for processes inspired by [14], we can drop neither the condition of cumulative fairness, nor remove self-loops completely (at least one on almost all of the nodes of the graph is necessary). If either of these assumptions is dropped, one can design processes which satisfy the fairness constraints of [14], yet have $\Omega(\operatorname{diam} d)$ discrepancy. Such discrepancy is much worse than the one given by Theorem 2.5 for many graph classes. Our lower bounds are presented in Section 5.

The next theorem shows that strictly fair algorithm achieve a smaller discrepancy of $O(d)$, but require a slightly longer time of $O\left(T + \frac{ds^{-1} \log^2 n}{\mu}\right)$ where we recall that $1 \leq s \leq (d^+ - d)$.

Theorem 2.6. *Let $1 \leq s \leq (d^+ - d)$ and let A be a strictly δ-fair algorithm which is s-selfish. Then A achieves $((2\delta + 1)d^+ + 4d^\circ)$-discrepancy after time*

$$O\left(\frac{\log K + ds^{-1} \cdot \log^2 n}{\mu}\right).$$

A proof of Theorem 2.6 is provided in Section 4.

We note that achieving high selfishness ($s = \Omega(d)$) increases the speed of the balancing process. However, since the selfishness is upper-bounded by the number of self-loops at a node, we need to impose $\Omega(d)$ self-loops to take advantage of this property.

Obtaining better local deterministic balancing schemes, i.e., achieving $o(d)$ discrepancy after a short time, appears to be infeasible. In Section 5, we provide lower bounds of $\Omega(d)$ on the discrepancy of stateless algorithms and one other specific process, the Rotor-Router*.

2.4 Examples of Algorithms and Specific Results

We consider two specific families of algorithms. One is the class of stateless algorithms called SEND, while the other consists of variants of the rotor-router model, described in the introduction. Herein we provide a formalization of these families, describing how the fairness and selfishness properties of these algorithms may be achieved, depending on their precise parametrization.

Stateless algorithms. We define the class of algorithms SEND$(g(x_t(u)))$ to be the class of algorithms for which every node $u \in V$ sends $g(x_t(u)) \in \mathbb{N}$ over every outgoing edge $e = (u, v) \in E_u$ with $v \in V$. All algorithms presented in this section fulfill $g(x_t(u)) \in \left\{ \left\lfloor \frac{x_t(u)}{d^+} \right\rfloor, \left\lceil \frac{x_t(u)}{d^+} \right\rceil \right\}$. Specific examples of SEND-class algorithms, which are also mentioned in Table 1, are given below:

1. SEND$\left(\left\lfloor \frac{x_t(u)}{d^+} \right\rfloor \right)$, for fixed $d^+ \geq 2d$. One can think of this algorithm as trying to distribute the tokens among all edges as fairly as possible such that every edge (and self-loop) receives the same number of tokens in every step. The tokens which cannot be distributed fairly are kept in the remainder vector. We observe that this algorithm is 0-cumulatively fair, but not strictly fair. Following Theorem 2.5, the achieved bound on discrepancy is $O\left(d \min\left\{\sqrt{\frac{\log n}{\mu}}, \sqrt{n}\right\}\right)$ in $O\left(\frac{\log(Kn)}{\mu}\right)$ time.

2. SEND$\left(\left\lceil \frac{x_t(u)}{2d} - \frac{1}{2} \right\rceil \right)$ This algorithm rounds on outgoing edges to the nearest integer which is either $\left\lceil \frac{x_t(u)}{2d} \right\rceil$ or $\left\lfloor \frac{x_t(u)}{2d} \right\rfloor$. This algorithm can be modelled by using d self-loops in the following way. Divide the sending process into two phases. In the first phase send to every neighbour $\left\lfloor \frac{x_t(u)}{d^+} \right\rfloor$ tokens. In the second phase do the following. Whenever $x_t(u) \bmod d^+ \leq d^+/2$, send no extra token to a neighbour. Otherwise, send one extra token to every neighbour. By encoding this process in our framework, we observe that this algorithm is 0-strictly fair and 1-selfish. Following Theorem 2.6, the algorithm achieves a $(10d)$-discrepancy in $O\left(\frac{\log K + d \log^2 n}{\mu}\right)$ time.

3. SEND$\left(\left\lceil \frac{x_t(u)}{3d} - \frac{2}{3} \right\rceil \right)$ This algorithm can be modelled similarly to the previous algorithm. This variant reaches a discrepancy close to $\bar{x}$ faster, but its discrepancy is slightly worse. The reason is that it has a selfishness of $\Omega(d)$ resulting in a faster balancing time. However, in order to have $\delta = 0$ and a selfishness of $\Omega(d)$, the algorithm requires $d + \Omega(d)$ self-loops. In this example, we use exactly $2d$ self-loops in the following way. Divide the sending process into two phases. In the first phase send to every neighbour $\left\lfloor \frac{x_t(u)}{d^+} \right\rfloor$ tokens. In the second phase do the following. Whenever $x_t(u) \bmod d^+ \leq 2d^+/3$, send no extra token to a neighbour. Otherwise send one extra token to every neighbour. We observe that this algorithm is 0-strictly fair and $\lceil d/3 \rceil$-selfish. Following Theorem 2.6, the algorithm achieves a $(17d)$-discrepancy in $O\left(\frac{\log K + \log^2 n}{\mu}\right)$ time.

Rotor-type algorithms. In the context of rotor-type algorithms, we consider the standard ROTOR-ROUTER approach with self-loops (for which we show the cumulative fairness property), as well as an algorithm we call ROTOR-ROUTER$*$, defined below, which includes a special self-loop to ensure the strict fairness and selfishness properties.

1. ROTOR-ROUTER This algorithm follows the standard rotor-router process on the balancing graph with added self-loops, where we assume for analysis that $d^+ \geq 2d$. The self-loops at a node u should be included in the cycle of the rotor when sending load to outgoing edges, at arbitrary positions with respect to the outgoing edges leading to a neighbour of u. The cyclicity property of the rotor-router guarantees that all edges at a node receive cumulatively the same number of tokens up to any time t, hence, ROTOR-ROUTER belongs to the class of 1-cumulatively fair algorithms. By Theorem 2.5, the ROTOR-ROUTER process admits a bound on discrepancy of $O\left(d \min\left\{\sqrt{\frac{\log n}{\mu}}, \sqrt{n}\right\}\right)$ after $O\left(\frac{\log(Kn)}{\mu}\right)$ time.

2. ROTOR-ROUTER∗ This algorithm differs from the standard rotor-router in that it maintains exactly one special self-loop, which always receives $\left\lceil \frac{x_t(u)}{2d} \right\rceil$ load. Additionally, there are $d-1$ self-loops introduced. The remaining load $(x_t(u) - \lceil x_t(u)/2d \rceil)$ is distributed fairly by using a rotor-router on the outgoing edges and the $d-1$ self-loops. This algorithm is 1-strictly fair and, due to the existence of the special self-loop, 1-selfish. By Theorem 2.6, it achieves a $(14d)$-discrepancy in $O\left(\frac{\log K + d\log^2 n}{\mu}\right)$ time. The bounds (i) and (ii) of Theorem 2.5 apply for this algorithm.

3. ROTOR-ROUTER∗SINGLELOOP This algorithm differs from the standard rotor-router in that it maintains exactly one special self-loop, which always receives $\left\lceil \frac{x_t(u)}{d+1} \right\rceil$ load. No additional self-loops are introduced in the balancing graph. The remaining load is distributed fairly by using a rotor-router on the outgoing edges. This algorithm is 1-strictly fair and, due to the existence of the special self-loop, 1-selfish. By Theorem 2.6, it achieves a $(5d+9)$-discrepancy in $O\left(\frac{\log K + d\log^2 n}{\mu}\right)$ time. The bounds (i) and (ii) of Theorem 2.5 do not apply for this algorithm since it doesn't have enough self-loops.

3 Cumulatively Fair Algorithms

In this section, we analyze the broad class of cumulatively fair load balancing algorithms, formally defined in Section 2, with the goal of proving Theorem 2.5.

The core idea is to regard the balancing process over several steps as opposed to focusing on one single step. By regarding a long period we will observe that the cumulative load, i.e., the total load which entered a node, on a node for our class of algorithms is closely related to a random walk of all tokens. The difference to the random walk can be bounded by a small corrective vector introducing a small error. This corrective vector depends on δ and r.

The obtained result will also be used as a starting point in the proof of Theorem 2.6, for the class of strictly fair and selfish algorithms. We will observe that the *time-averaged* load of u for any cumulatively fair algorithm, where the averaging takes places over an interval of consecutive time steps. The difference between these two values depends on the unfairness δ, the bound on the remainder r, and a term which is diminishingly small if the period considered is sufficiently long, as we will see later.

Before continuing with the proof, we require some further definitions.

3.1 Preliminaries

Random walks. Let $\mathbf{P}$ denote the transition matrix of a random walk of G^+. Let $\mathbf{P}(u,v)$ denote the one-step probability for the walk to go from u to v. Then

$$\mathbf{P}(u,v) = \begin{cases} 1/d^+ & (u,v) \in E \\ d^\circ/d^+ & u = v \\ 0 & \textit{otherwise} \end{cases}.$$

Let μ be the eigenvalue gap of P. We define $\mathbf{P}^t$ to be the t-steps transition matrix, i.e., $\mathbf{P}^t = \mathbf{P} \cdot \mathbf{P}^{t-1}$. We define the *steady-state* distribution as $\mathbf{P}^\infty = \lim_{t\to\infty} \mathbf{P}^t$. Note that $\forall u, v \in V$, $\mathbf{P}^\infty(u,v) = d^+/(2|E^+|) = 1/n$. Observe that $P^\infty \cdot \mathbf{x}_1 = (\bar{x}, \bar{x}, \ldots, \bar{x})$. We can express $\mathbf{P}^t$ as $\mathbf{P}^t = \mathbf{P}^\infty + \mathbf{\Lambda}_t$, where $\mathbf{\Lambda}_t$ is the error-term calculating the difference between $\mathbf{P}^t$ and the steady-state distribution. We define similar to [12]

$$\text{diff}(t) = \max_{\|\mathbf{q}\|_1 = 1} \{\left\|(\mathbf{P}^t - \mathbf{P}^\infty)\mathbf{q}\right\|_1\} = \max_{\|\mathbf{q}\|_1 = 1} \{\|\mathbf{\Lambda}_t \mathbf{q}\|_1\}.$$

The mixing time t_{mix} is defined as $\arg\min_t\{\text{diff}(t) \le 1/2\}$.

Vector norm. Let $p \ge 1$. The p-norm of a vector $\mathbf{r}$ is defined as

$$\|\mathbf{r}\|_p = \left(\sum_{i=1}^{n} |r_i|^p\right)^{\frac{1}{p}}.$$

In particular, $\|\mathbf{r}\|_\infty$ is defined to be $\max\{r_1, \ldots, r_n\}$.
We start with a technical lemma giving bounds on the behaviour of the error matrix $\mathbf{\Lambda}_\tau$ for $\tau \in \mathbb{N}$ with $\tau \geq t_{mix}$. We recall that $\mathbf{\Lambda}_\tau$ measures the difference between the steady-state distribution and the transition matrix after τ steps, i.e., P^τ.

Lemma 3.1 ([12]). *Let $(\mathbf{q}_t)$ be a sequence of vectors parameterized by t, let $\bar{\mathbf{q}}_t = \mathbf{P}^\infty \mathbf{q}_t$, and let $c \in \mathbb{N}$ be an arbitrary constant.*

(i) For $t \geq c \cdot 4\frac{\log(n \cdot \max_\tau \|\mathbf{q}_\tau - \bar{\mathbf{q}}_\tau\|_\infty)}{\mu}$ we have

$$\|\mathbf{\Lambda}_t \mathbf{q}_t\|_\infty \leq 2^{-c} \tag{3}$$

(ii) Let t_{mix} be the mixing time, $t_{mix} = O\left(\frac{\log n}{\mu}\right)$, then

$$\sum_{t \geq c \cdot t_{mix}} \|\mathbf{\Lambda}_t \mathbf{q}_t\|_\infty \leq 2^{-c+1} \cdot \max_{\tau \geq c \cdot t_{mix}} \{\|\mathbf{q}_\tau\|_\infty\} \tag{4}$$

3.2 Key Lemma: Approaching the Average Load

The following lemma shows that, starting from some minimum time $t = O(T)$, the load of every node in a cumulative diffusion process must decrease, at least for one time step, to a value not far from the average load $\bar{x}$ of the system. The time after which this happens follows precisely from the Lemma, but can in general be bounded as $O(T)$.

Lemma 3.2. *Consider any cumulatively δ-fair algorithm with remainder bounded by r, and an initialization of the load balancing process with average load $\bar{x}$ and initial discrepancy K. Let $\lambda \geq 0$, let $t \geq 16\frac{\log(nK)}{\mu}$, and let $\widehat{T} = O\left(\frac{d \log n}{\mu(\lambda+1)}\right)$. We have:*

For all $u \in V$ there exists a time step $t' \in [t+1; t+\widehat{T}]$ such that $x_{t'}(u) \leq \bar{x} + \delta d^+ + 2r + \frac{1}{2} + \lambda$.

Proof. Fix a node $u \in V$. Note that by the definition of cumulative δ-fairness we have

$$\forall_{(u,v) \in E_u^+} \left| F_t(u,v) - \frac{F_t^{out}(u)}{d^+} \right| \leq \delta \tag{5}$$

Let $\boldsymbol{\delta}_{t,u}$ be a vector parameterized by $u \in V$ and time $t \in \mathbb{N}$. We derive from (5)

$$\begin{aligned}
F_t^{in}(u) &\underset{def.}{=} \sum_{v \in N(u)} F_t(v,u) \\
&= \sum_{v \in N(u) \setminus \{u\}} F_t(v,u) + F_t(u,u) \\
&\underset{(5)}{=} \sum_{v \in N(u) \setminus \{u\}} \left(\frac{1}{d^+} F_t^{out}(v) + \delta_{t,u}(v) \right) + \frac{d^\circ}{d^+} F_t^{out}(u) + \delta_{t,u}(u) \\
&= \sum_{v \in N(u) \setminus \{u\}} \frac{1}{d^+} F_t^{out}(v) + \frac{d^\circ}{d^+} F_t^{out}(u) + \|\boldsymbol{\delta}_{t,u}\|_1 .
\end{aligned} \tag{6}$$

Hence, $\boldsymbol{\delta}_{t,u}$ can be seen as a corrective vector satisfying $|\delta_{t,u}(v)| \leq \delta$ for $v \in N(u) \setminus \{u\}$, $|\delta_{t,u}(u)| \leq d^\circ \delta$, and $\delta_{t,u}(v) = 0$ for $v \notin N(u)$. Consequently, $\|\boldsymbol{\delta}_{t,u}\|_1 \leq \delta d^+$. Rewriting (1) by introducing (6) we get:

$$F_t^{out}(u) = x_1(u) + \sum_{v \in N(u) \setminus \{u\}} \frac{1}{d^+} F_{t-1}^{out}(v) + \frac{d^\circ}{d^+} F_{t-1}^{out}(u) + \underbrace{\|\boldsymbol{\delta}_{t,u}\|_1 - r_t(u)}_{\varepsilon_t(u)} \tag{7}$$

We have $\|\varepsilon_t(u)\|_\infty \le \delta d^+ + r$. Rewriting (7) in vector form, we obtain

$$\mathbf{F}_t^{out} = \mathbf{x}_1 + \mathbf{P} \cdot \mathbf{F}_{t-1}^{out} + \boldsymbol{\varepsilon}_t.$$

Expanding with respect to t gives

$$\mathbf{F}_t^{out} = \sum_{0\le\tau<t} \mathbf{P}^\tau \mathbf{x}_1 + \sum_{1\le\tau\le t} \mathbf{P}^{t-\tau}\boldsymbol{\varepsilon}_\tau.$$

For any $\widehat{T} > 0$, the number of tokens which left nodes in the interval of steps $[t+1; t+\widehat{T}]$ is

$$\mathbf{F}_{t+\widehat{T}}^{out} - \mathbf{F}_t^{out} = \sum_{t\le\tau<t+\widehat{T}} \mathbf{P}^\tau \mathbf{x}_1 + \sum_{1\le\tau\le t} (\mathbf{P}^{t+\widehat{T}-\tau} - \mathbf{P}^{t-\tau})\boldsymbol{\varepsilon}_\tau + \sum_{t<\tau\le t+\widehat{T}} \mathbf{P}^{t+\widehat{T}-\tau}\boldsymbol{\varepsilon}_\tau.$$

Fixing $t^* = t - 4t_{mix}$ and substituting $\mathbf{P}^\tau = \mathbf{P}^\infty + \boldsymbol{\Lambda}_\tau$:

$$\begin{aligned}
\mathbf{F}_{t+\widehat{T}}^{out} - \mathbf{F}_t^{out} &= \sum_{t\le\tau<t+\widehat{T}} \mathbf{P}^\tau \mathbf{x}_1 + \sum_{1\le\tau\le t^*} (\mathbf{P}^{t+\widehat{T}-\tau} - \mathbf{P}^{t-\tau})\boldsymbol{\varepsilon}_\tau + \sum_{t^*<\tau\le t} (\mathbf{P}^{t+\widehat{T}-\tau} - \mathbf{P}^{t-\tau})\boldsymbol{\varepsilon}_\tau + \sum_{t<\tau\le t+\widehat{T}} \mathbf{P}^{t+\widehat{T}-\tau}\boldsymbol{\varepsilon}_\tau \\
&= \widehat{T}\mathbf{P}^\infty \mathbf{x}_1 + \sum_{t\le\tau<t+\widehat{T}} \boldsymbol{\Lambda}_\tau \mathbf{x}_1 \\
&\quad + \sum_{1\le\tau\le t^*} (\boldsymbol{\Lambda}_{t+\widehat{T}-\tau} - \boldsymbol{\Lambda}_{t-\tau})\boldsymbol{\varepsilon}_\tau + \sum_{t^*<\tau\le t} (\mathbf{P}^{t+\widehat{T}-\tau} - \mathbf{P}^{t-\tau})\boldsymbol{\varepsilon}_\tau + \sum_{t<\tau\le t+\widehat{T}} \mathbf{P}^{t+\widehat{T}-\tau}\boldsymbol{\varepsilon}_\tau. \qquad (8)
\end{aligned}$$

Therefore, the maximum difference taken over all nodes between the average load of the node in the last $\widehat{T}$ time steps and the average load $\bar{x}$, after multiplying by $\widehat{T}$ is

$$\begin{aligned}
\left\| \sum_{t<\tau\le t+\widehat{T}} \mathbf{x}_\tau - \widehat{T}\cdot\bar{\mathbf{x}} \right\|_\infty &\underset{(2)}{=} \left\| \left(\mathbf{F}_{t+\widehat{T}}^{out} - \mathbf{F}_t^{out} + \sum_{t<\tau\le t+\widehat{T}} \mathbf{r}_\tau \right) - \widehat{T}\mathbf{P}^\infty \mathbf{x}_1 \right\|_\infty \\
&\underset{(8)}{\le} \sum_{t\le\tau<t+\widehat{T}} \|\boldsymbol{\Lambda}_\tau \mathbf{x}_1\|_\infty \\
&\quad + \sum_{1\le\tau\le t^*} \left(\left\|\boldsymbol{\Lambda}_{t+\widehat{T}-\tau}\boldsymbol{\varepsilon}_\tau\right\|_\infty + \|\boldsymbol{\Lambda}_{t-\tau}\boldsymbol{\varepsilon}_\tau\|_\infty \right) + \sum_{t^*<\tau\le t} \left\|(\mathbf{P}^{t+\widehat{T}-\tau} - \mathbf{P}^{t-\tau})\boldsymbol{\varepsilon}_\tau\right\|_\infty \\
&\quad + \sum_{t<\tau\le t+\widehat{T}} \left\|\mathbf{P}^{t+\widehat{T}-\tau}\boldsymbol{\varepsilon}_\tau\right\|_\infty \\
&\quad + \sum_{t<\tau\le t+\widehat{T}} \|\mathbf{r}_\tau\|_\infty
\end{aligned}$$

($t \ge 16\frac{\log(nK)}{\mu}$ by (3) implies that $\forall_{\tau\ge t} \|\boldsymbol{\Lambda}_\tau \mathbf{x}_1\|_\infty \le 2^{-4}$, and (4) implies that $\sum_{\tau\ge 4\cdot t_{mix}} \|\boldsymbol{\Lambda}_\tau\boldsymbol{\varepsilon}_\tau\|_\infty \le 2^{-3}\cdot \max_{\tau\ge 4\cdot t_{mix}}\{\|\boldsymbol{\varepsilon}_\tau\|_\infty\}$). Next we have

$$\begin{aligned}
\left\| \sum_{t<\tau\le t+\widehat{T}} \mathbf{x}_\tau - \widehat{T}\cdot\bar{\mathbf{x}} \right\|_\infty &\le \widehat{T}\cdot 2^{-4} \\
&\quad + 2\cdot 2^{-3} \max_{1\le\tau\le t^*} \|\boldsymbol{\varepsilon}_\tau\|_\infty + \sum_{t^*<\tau\le t} \left\|(\mathbf{P}^{t+\widehat{T}-\tau} - \mathbf{P}^{t-\tau})\boldsymbol{\varepsilon}_\tau\right\|_\infty \\
&\quad + \widehat{T}(\delta d^+ + r) \\
&\quad + \widehat{T} r \\
&\le \frac{\widehat{T}}{4} + (\delta d^+ + r) + \sum_{t^*<\tau\le t} \left\|(\mathbf{P}^{t+\widehat{T}-\tau} - \mathbf{P}^{t-\tau})\boldsymbol{\varepsilon}_\tau\right\|_\infty + \widehat{T}(\delta d^+ + 2r). \qquad (9)
\end{aligned}$$

Dividing (9) by $\widehat{T}$ yields

$$\left\| \frac{\sum_{t<\tau\leq t+\widehat{T}} \mathbf{x}_\tau}{\widehat{T}} - \bar{\mathbf{x}} \right\|_\infty \leq \frac{1}{4} + (\delta d^+ + 2r) + \frac{(\delta d^+ + r) + \sum_{t^*<\tau\leq t} \left\| (\mathbf{P}^{t+\widehat{T}-\tau} - \mathbf{P}^{t-\tau})\boldsymbol{\varepsilon}_\tau \right\|_\infty}{\widehat{T}}. \tag{10}$$

We bound the term $\sum_{t^*<\tau\leq t} \left\| (\mathbf{P}^{t+\widehat{T}-\tau} - \mathbf{P}^{t-\tau})\boldsymbol{\varepsilon}_\tau \right\|_\infty$ of (10):

$$\begin{aligned}
\sum_{t^*<\tau\leq t} \left\| (\mathbf{P}^{t+\widehat{T}-\tau} - \mathbf{P}^{t-\tau})\boldsymbol{\varepsilon}_\tau \right\|_\infty &\leq \sum_{t^*<\tau\leq t} \left(\left\| (\mathbf{P}^{t+\widehat{T}-\tau}\boldsymbol{\varepsilon}_\tau \right\|_\infty + \left\| \mathbf{P}^{t-\tau}\boldsymbol{\varepsilon}_\tau \right\|_\infty \right) \\
&\leq 2 \sum_{t^*<\tau\leq t} \|\boldsymbol{\varepsilon}_\tau\|_\infty \\
&\leq 2(\delta d^+ + r)(t - t^*) = 8t_{mix}(\delta d^+ + r).
\end{aligned}$$

Injecting this into the (10) and dividing by $\widehat{T}$ gives:

$$\left\| \frac{\sum_{t<\tau\leq t+\widehat{T}} \mathbf{x}_\tau}{\widehat{T}} - \bar{x} \right\|_\infty \leq \left(\delta d^+ + 2r + \frac{1}{4} \right) + \frac{(8t_{mix} + 1)(\delta d^+ + r)}{\widehat{T}}. \tag{11}$$

Since $d^\circ = O(d^+)$, $\delta = O(1)$, and $r \leq d^+$, there is constant c such that $cd \geq \delta d^+ + r$. By [12], we have $t_{mix} \leq c' \frac{\log n}{\mu}$ for some constant c'. We set $\widehat{T} \geq 36c \cdot c' \cdot \frac{d \log n}{\mu(\lambda+1)}$.
We derive from (11)

$$\left\| \frac{\sum_{t<\tau\leq t+\widehat{T}} \mathbf{x}_\tau}{\widehat{T}} - \bar{x} \right\|_\infty \leq \delta d^+ + 2r + \frac{1}{4} + \frac{9t_{mix} \cdot cd}{36c \cdot c' \cdot \frac{d \log(n)}{\mu(\lambda+1)}} \leq \delta d^+ + 2r + \frac{1}{4} + \frac{(\lambda+1)}{4} \leq \delta d^+ + 2r + \frac{1}{4} + \lambda + \frac{1}{4}.$$

Therefore, we have that the difference between the average load of any node over $\widehat{T} = \frac{c' d \log(n)}{\mu(\lambda+1)}$ steps and the average load $\bar{x}$ is for bounded by $\delta d^+ + 2r + \frac{1}{2} + \lambda$. This means that for every node u there has to be a time step $t' \in [t+1, t+\widehat{T}]$ such that $x_{t'}(u) - \bar{x} \leq \delta d^+ + 2r + \frac{1}{2} + \lambda$. This yields the claim. □

We note that a bound symmetric to the above Lemma can be established for approaching the average load $\bar{x}$ from below with a cumulatively fair process.

We are ready to prove the discrepancy bounds in Theorem 2.5.

3.3 Proof of Theorem 2.5

We recall the statement of the Theorem 2.5.

Theorem 2.5 Let A be any cumulatively fair algorithm. For any d-regular input graph G, after $O\left(\frac{\log(Kn)}{\mu}\right)$ time, A achieves:

(i) $O\left(d \cdot \sqrt{\frac{\log n}{\mu}}\right)$-discrepancy for $d^+ \geq 2d$.

(ii) $O\left(d \cdot \sqrt{n}\right)$-discrepancy for $d^+ \geq 2d$.

(iii) $O\left(d \cdot \frac{\log n}{\mu}\right)$-discrepancy for arbitrary $d^+ \geq d + 1$.

where d^+ is the degree of the balancing graph (including self-loops) used by A.

Proof. In the following, we fix an arbitrary $t \geq 16\frac{\log(nK)}{\mu}$. Let w be the i'th node of V, then we define $\mathbf{w}$ to be the vector, such that $\mathbf{w}[i] = 1$ and $\forall_{j \neq i}\mathbf{w}[j] = 0$. We define $P_t(u, w)$ to be the probability that a random walk following matrix $\mathbf{P}$, initially located at $u \in V$, is located at w after t time steps.

We observe, that by fixing $\widehat{T} = 1$, (10) bounds on: $\left\|\frac{\sum_{t<\tau\leq t+\widehat{T}}\mathbf{x}_\tau}{\widehat{T}} - \bar{\mathbf{x}}\right\|_\infty = \|\mathbf{x}_{t+1} - \bar{\mathbf{x}}\|_\infty$, the load discrepancy.

We start by bounding the term $\left\|(\mathbf{P}^{t+1-\tau} - \mathbf{P}^{t-\tau})\boldsymbol{\varepsilon}_\tau\right\|_\infty$ of (10) where $t^* < \tau \leq t$, denoting $a = t - \tau$ $(0 \leq a < t - t^*)$:

$$\begin{aligned}
\left\|(\mathbf{P}^{t+1-\tau} - \mathbf{P}^{t-\tau})\boldsymbol{\varepsilon}_\tau\right\|_\infty &= \left\|(\mathbf{P}^{a+1} - \mathbf{P}^{a})\boldsymbol{\varepsilon}_{t-a}\right\|_\infty \\
&= \max_{w\in V}\left|\mathbf{w}^\top\left(\mathbf{P}^{a+1} - \mathbf{P}^{a}\right)\boldsymbol{\varepsilon}_{t-a}\right| \\
&= \max_{w\in V}\left|\mathbf{w}^\top\left(\mathbf{P}^{a+1} - \mathbf{P}^{a}\right)\left(\sum_{v\in V}\boldsymbol{\varepsilon}_{t-a}(v)\mathbf{v}\right)\right| \\
&= \max_{w\in V}\left|\sum_{v\in V}\boldsymbol{\varepsilon}_{t-a}(v)\left(\mathbf{w}^\top\left(\mathbf{P}^{a+1} - \mathbf{P}^{a}\right)\mathbf{v}\right)\right| \\
&\leq \|\boldsymbol{\varepsilon}_{t-a}\|_\infty \cdot \max_{w\in V}\sum_{v\in V}\left|\mathbf{w}^\top(\mathbf{P}^{a+1} - \mathbf{P}^{a})\mathbf{v}\right| \\
&\leq (\delta d^+ + r)\cdot\max_{w\in V}\sum_{v\in V}|P_{a+1}(v,w) - P_a(v,w)| \qquad (12)
\end{aligned}$$

Then, since the graph is regular, we have $P_t(v, w) = P_t(w, v)$.[1] From here on we divide the analysis and prove the claimed bounds separately.

(i) $O\left(d\sqrt{\frac{\log n}{\mu}}\right)$**-discrepancy for** $d^+ \geq 2d$ **:**
For regular graphs having $P(u, u) \geq 1/2$ for all $u \in V$, we have by [12] (for $a > 0$)

$$\forall_{w\in V}\sum_{v\in V}|P_{a+1}(w,v) - P_a(w,v)| < \frac{24}{\sqrt{a}}.$$

(For case of $a = 0$, we have $\forall_{w\in V}\sum_{v\in V}|P_1(w,v) - P_0(w,v)| \leq 2$.)

We obtain by applying $\sum_{a=1}^{t-t^*} 1/\sqrt{a} \leq 2\sqrt{t - t^*}$

$$\sum_{0\leq a<t-t^*}\left\|(\mathbf{P}^{a+1} - \mathbf{P}^{a})\boldsymbol{\varepsilon}_{t-a}\right\|_\infty < (\delta d^+ + r)\left(2 + 48\sqrt{\underbrace{t - t^*}_{4t_{mix}}}\right) \leq 98(\delta d^+ + r)\sqrt{t_{mix}}.$$

Introducing this into into the (10) and setting $\widehat{T} = 1$ yields

$$\|\mathbf{x}_{t+1} - \bar{x}\|_\infty = O\left((\delta d^+ + r)\sqrt{t_{mix}}\right) = O\left(((\delta + 1)d)\sqrt{\frac{\log n}{\mu}}\right)$$

where the last equality follows from $t_{mix} = O(\frac{\log n}{\mu})$ (see [12]) and from $r \leq d^+$. (Observe that whenever an cumulatively fair algorithm has a $\delta = 0$ the bound on the remainder r has to be of order $\Omega(d)$.)

(ii) $O\left(d\sqrt{n}\right)$**-discrepancy for** $d^+ \geq 2d$ **:**

[1] For general graphs, one can use $P_t(v, w) = (d^+(w)/d^+(v))P_t(w, v)$

Let D_{a+1} be the diagonal matrix of $(P_{a+1} - P_a)$ and let X be the corresponding base change matrix.

$$\begin{aligned}
\max_{w\in V} \sum_{v\in V} |P_{a+1}(w,v) - P_a(w,v)| &= \max_{w\in V} \sum_{v\in V} |\mathbf{v}(P_{a+1} - P_a)\mathbf{w}| \\
&= \max_{w\in V} \|(P_{a+1} - P_a)\mathbf{w}\|_1 \\
&\leq \max_{\|\mathbf{w}\|_2=1} \|(P_{a+1} - P_a)\mathbf{w}\|_1 \\
&= \max_{\|\mathbf{w}\|_2=1} \left\|X^\top D_{a+1} X\mathbf{w}\right\|_1 \\
&\leq \sqrt{n} \left\|X^\top D_{a+1} X\right\|_2 \\
&= \sqrt{n} \|D_{a+1}X\|_2 . \qquad (13)
\end{aligned}$$

We have

$$D_{a+1} = \begin{pmatrix} \lambda_1^{a+1} - \lambda_1^a & 0 & \dots & 0 \\ 0 & \lambda_2^{a+1} - \lambda_2^a & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n^{a+1} - \lambda_n^a \end{pmatrix}.$$

where $\lambda_1, \dots, \lambda_n$ are the eigenvalues of P. We note that $\lambda_1 = 1$ and $\lambda_2, \dots, \lambda_n \in [0,1]$ since $d^\circ \geq d$. Hence by plugging (13) in (12) we derive

$$\begin{aligned}
\sum_{0\leq a<t-t^*} \left\|(\mathbf{P}^{t+\widehat{T}-\tau} - \mathbf{P}^{t-\tau})\boldsymbol{\varepsilon}_\tau\right\|_\infty &\leq (\delta+1)d^+ \cdot \sum_{a<t-t^*} \max_{w\in V} \sum_{v\in V} |P_{a+1}(w,v) - P_a(w,v)| \\
&\leq (\delta+1)d^+ \cdot \sum_{a<t-t^*} \sqrt{n} \|D_{a+1}X\|_2 \\
&\leq (\delta+1)d^+ \cdot \sqrt{n} \sum_{a=0}^{t-t^*} |\lambda_2^{a+1} - \lambda_2^a| \\
&= (\delta+1)d^+ \cdot \sqrt{n} \cdot (\lambda_2^0 - \lambda_2^{t-t^*}) \\
&\leq (\delta+1)d^+ \cdot \sqrt{n}
\end{aligned}$$

Introducing this into (10) and setting $\widehat{T} = 1$ yields:

$$\|\mathbf{x}_{t+1} - \bar{x}\|_\infty \leq (\delta+1)d^+ \cdot \sqrt{n},$$

which completes the proof.

(iii) $O\left(\frac{d \log n}{\mu}\right)$**-discrepancy for** $d^+ \geq d+1$**:**
Follows from (11) by setting $\widehat{T} = 1$.

□

4 Analysis of Strictly Fair and Selfish Algorithms

In this section, we provide a sharper bound on discrepancy for algorithms which are strictly fair and selfish. To this end, we will make use of the linear algebra-based results from the previous section, and combine them with the following combinatorial potential-based approach. The introduced potentials are related to levels of load of nodes, arranged at multiples of d^+. Within the considered class of algorithms, whenever all nodes have a load under a threshold, their load will remain under this threshold (see Lemma 4.1(i)). Moreover, one can define a pair of potentials, one of which decreases whenever a node moves from over some threshold to under it, and the other — whenever a node moves from under a threshold to over it (see Lemma 4.1 and Lemma 4.3, respectively). Using these potentials is the main idea in this section. In particular, we can use

Lemma 3.2 to show that nodes having a load over a certain threshold (having a certain distance to $\bar{x}$) will, at some point within the next $O(\log(Kn)/\mu)$ rounds, drop to a load below the threshold. This enforces a drop of one of the mentioned potentials. Eventually, all nodes will have load concentrated around the average load $\bar{x}$ of the system, and the considered strictly fair selfish algorithm will achieve $O(d)$-discrepancy. The exact time after which this happens follows from the proof of Theorem 2.6.

4.1 Lemmas on Potential Drop

Lemma 4.1 (Monotonicity of potential)**.** *Let A be strictly fair and selfish. For $c \in \mathbb{N}$, we define the following family of potentials parameterized by c:*

$$\phi_t(c) = \sum_{v \in V} \max\{x_t(v) - cd^+, 0\}.$$

The potential $\phi_t(c)$

(i) is non-increasing in time.

(ii) satisfies: $\phi_t(c) \leq \phi_{t-1}(c) - \sum_{u \in V} \Delta_t(c,u)$, where:

$$\Delta_t(c,u) = \begin{cases} \min\{x_{t-1}(u), cd^+ + s\} - \max\{x_t(u), cd^+\} & \text{if } x_{t-1}(u) > x_t(u) \text{ and } x_{t-1}(u) > cd^+ \\ & \text{and } x_t(u) < cd^+ + s \\ 0 & \text{otherwise} \end{cases}$$

Before proving the lemma, we remark that in clause (ii), the potential drops for any algorithm which is at least 1-selfish, at time t (i.e., $\Delta_t(c,u) \geq 1$) for any node u such that $x_{t-1}(u) \geq cd^+ + 1$ and $x_t(u) \leq cd^+$.

Proof. Fix $c \in \mathbb{N}$. At any time t, we will divide the set L of m tokens circulating in the system into two groups: the set of *black* tokens L_t^- and the set of *red* tokens L_t^+, with $L = L_t^- \cup L_t^+$. Colors of tokens persist over time unless they are explicitly recolored. For $t = 1$, for each node u we color exactly $|L_1^-(u)| = \min\{x_1(u), cd^+\}$ tokens at u black, and the remaining tokens at u red. In every time step, we follow two rules concerning token distribution:

(1) The number of black tokens leaving a node u along any edge (including self-loops) is never more than c.

(2) At the start of each subsequent step t, we recolor some red tokens to black, so that the total number of black tokens located at a node u is exactly $|L_t^-(u)| = \min\{x_t(u), cd^+\}$.

We note that both rules of token circulation are well defined. The proof proceeds by induction. To prove the correctness of rule (1) at step t, observe that, by the definition of strictly fair algorithms, for any node u we either have $x_t(u) \leq cd^+$ and then node u sends at most c tokens along each of its edges and self-loops, or $x_t(u) > cd^+$, and then node u sends at least c tokens along each of its edges and self-loops. In the first case, rule (1) is correct regardless of how u distributes tokens of different colors; in the second case, u has exactly cd^+ black tokens, and we can require that it sends exactly c of its black tokens along each of its edges and self-loops. To prove the correctness of rule (2), we note that by the correctness of rule (1) for the preceding time step, the number of black tokens arriving at u along edges and self-loops can be upper-bounded by $\min\{x_t(u), cd^+\}$. Hence, no recoloring of tokens from black to red is ever required.

We now observe that the potential $\phi_t(c)$ is by definition the number of red tokens circulating in the system at any given moment of time. Indeed, we have:

$$\phi_t(c) = \sum_{u \in V} (x_t(u) - \min\{x_t(u), cd^+\}) = m - \sum_{u \in V} |L_t^-(u)| = m - |L_t^-| = |L_t^+|.$$

The monotonicity condition (i) for the potential follows immediately from the fact that no new red tokens appear in the system. To prove (ii), we will show that the number of tokens being recolored from red to black at node u in step t is at least $\Delta_t(c,u)$. Indeed, suppose that at time $t-1$ we had for some node u:

$x_{t-1}(u) = cd^+ + i$, for some $i \geq 1$. Then, by definition of the selfishness of algorithm A, at least $c+1$ units of load will be sent on at least $i' = \min\{i, s\}$ self-loops of u in step t. By rule (1) of the token circulation process, each of these self-loops will contain at least one red token. Thus, the number of red tokens arriving at u at time t is at least i'. On the other hand, the number of red tokens remaining after the recoloring at u in step t is precisely $\max\{x_t(u) - cd^+, 0\}$. Thus, the number of tokens recolored from red to black at u, or equivalently the potential drop induced at u, is at least

$$\max\{i' - \max\{x_t(u) - cd^+, 0\}, 0\} = \max\{\min\{x_{t-1}(u) - cd^+, s\} - \max\{x_t(u) - cd^+, 0\}, 0\} \stackrel{\text{def}}{=} \Delta_t(c, u)$$

which completes the proof of (ii). □

The following observation follows directly from Lemma 4.1; we omit its proof.

Lemma 4.2. *Let A be an s-selfish algorithm and let $t \leq t'$ be two fixed time steps. Denote by U the subset of nodes such that for all $u \in U$ $x_t(u) \geq cd^+ + 1$ and there exists a moment of time $t_u \in [t, t']$ such that $x_{t_u}(u) \leq cd^+$. Then*

$$\phi_{t'}(c) \leq \phi_t(c) - \sum_{u \in U} \max\{s, x_t(u) - cd^+\}.$$

□

The potential defined in Lemma 4.1 bounds the number of tokens above certain thresholds. Now we present a symmetric potential measuring the number of 'gaps' below certain thresholds.

Lemma 4.3. *Let A be strictly fair and selfish. For $c \in \mathbb{N}$, we define the following family of potentials parameterized by c:*

$$\phi'_t(c) = \sum_{v \in V} \max\{cd^+ + s - x_t(v), 0\}.$$

The potential $\phi'_t(c)$

(i) is non-increasing in time.

(ii) satisfies: $\phi'_t(c) \leq \phi'_{t-1}(c) - \sum_{u \in V} \Delta'_t(c, u)$, where:

$$\Delta'_t(c,u) = \begin{cases} \min\{x_t(u), cd^+ + s\} - \max\{x_{t-1}(u), cd^+\} & \text{if } x_{t-1}(u) < x_t(u) \text{ and } x_{t-1}(u) < cd^+ + s \\ & \quad \text{and } x_t(u) > cd^+ \\ 0 & \text{otherwise} \end{cases}$$

Before proving the lemma, we remark that in clause (ii), the potential admits a drop at node u at time t (i.e., $\Delta'_t(c, u) \geq 1$) for every node u such that $x_{t-1}(u) \leq cd^+$ and $x_t(u) \geq cd^+ + 1$, for any algorithm which is at least 1-selfish.

The proof of Lemma 4.3 is very similar to that of Lemma 4.1, and we provide it for completeness in the Appendix.

The following observation follows directly from Lemma 4.3; we omit its proof.

Lemma 4.4. *Let A be an s-selfish algorithm and let $t \leq t'$ be two fixed time steps. Denote by U the subset of nodes such that for all $u \in U$ $x_t(u) < cd^+ + s$ and there exists a moment of time $t_u \in [t, t']$ such that $x_{t_u}(u) \geq cd^+ + s$. Then*

$$\phi'_{t'}(c) \leq \phi_t(c) - \sum_{u \in U} \max\{s, cd^+ + s - x_t(u)\}.$$

□

We are now ready to prove the main theorem of this section.

4.2 Proof of Theorem 2.6

We recall the statement of the Theorem.

Theorem 2.6. Let $1 \leq s \leq d^\circ$ and let A be a strictly δ-fair algorithm which is s-selfish. Then A achieves $((2\delta+1)d^+ + 4d^\circ)$-discrepancy after time

$$O\left(\frac{\log K + ds^{-1} \cdot \log^2 n}{\mu}\right).$$

Proof. We consider the behaviour of the process for $t \geq T = O(\frac{\log(Kn)}{\mu})$. Since the maximum load cannot increase in time, from this time moment onwards, the maximum load of a node will not exceed $x_m = \bar{x} + O\left(\frac{d \log n}{\mu}\right)$ by Theorem 2.5(iii).

In the first part of the proof, we will show that after a further $O(\frac{d^+}{s}\frac{\log^2 n}{\mu})$ time steps, the maximum load in the network will drop below the threshold value $c_0 d^+$, where c_0 is the smallest integer such that $c_0 d^+ \geq \bar{x} + \delta d^+ + 2d^\circ + d^+/2$.

We note that if there exists a node u with load $x_t(u) \geq c_0 d^+ + 1$ at some time moment $t \geq T$, then by Lemma 3.2 with $\lambda = d^+/2 - 1/2$ (which we can apply to strictly δ-fair algorithms, putting $r = d^\circ$, by Proposition 2.3), there exists some time step $t' \in [t+1; t+\widehat{T}]$, where $\widehat{T} = O\left(\frac{\log n}{\mu}\right)$, such that we have $x_{t'}(u) \leq c_0 d^+$. We then obtain directly from Lemma 4.2 that such a load change for node u results in the decrease of potential $\phi(c_0)$ in the time interval $[t+1; t+\widehat{T}]$. Since the potential $\phi(c_0)$ is non-increasing and non-negative, it follows that eventually the load of all nodes must be below the threshold $c_0 d^+$.

In order to prove that the load of all nodes drop below $c_0 d^+$ within $O(\frac{d^+}{s}\frac{\log^2 n}{\mu})$ time steps after time T, we apply a more involved potential-decrease argument based on the parallel drop of multiple potentials $\phi(c)$, for $c \in \{c_0, c_0+1, \ldots, c_1\}$. Here, c_1 is the smallest integer such that $c_1 d^+ \geq x_m = \bar{x} + O\left(\frac{d \log n}{\mu}\right)$.

We now partition the execution of our process into phases of duration t_p, $p = 1, 2, 3, \ldots, p_f$, such that, at the end of moment $T_p = T + t_1 + \ldots + t_p$ (end of the p'th phase), the following condition is satisfied:

$$\phi_{T_p}(c) \leq 4^{(c_1-c)} 2^{-p} (c_1 - c_0) d^+ n, \quad \text{for all } c_0 \leq c \leq c_1. \tag{14}$$

As we have remarked, the values of time t_p are well defined, since eventually the potentials $\phi(c)$ drop to 0, for all $c_0 \leq c \leq c_1$. Our goal is now to bound the ending time T_{p_f} of phase p_f, where:

$$p_f = 2(c_1 - c_0) + \lceil \log((c_1 - c_0) d^+ n) \rceil + 1. \tag{15}$$

At the end of this phase, we will have by (14) that $\phi_{T_{p_f}}(c_0) \leq 1/2$, hence $\phi_{T_{p_f}}(c_0) = 0$, and so there are no nodes having load exceeding $c_0 d^+$.

We now proceed to show the following bounds on the duration of each phase t_p:

$$t_p = O\left(\frac{d^+}{s} \frac{d \log n}{\mu \cdot \max\{(2(c_1 - c_0) - p)d^+ + 1, d^+/2 + 1\}}\right). \tag{16}$$

For any fixed p, assume that bound (16) holds for all phases before p. For phase p, the proof proceeds by induction with respect to c, in decreasing order of values: $c = c_1, c_1 - 1, \ldots, c_0$.

First, we consider values of $c \geq c_1 - p/2$. For a fixed c, following (14) we denote $b = 4^{(c_1-c)} 2^{-p} (c_1 - c_0) d^+ n$. Knowing that $\phi_{T_{p-1}}(c) \leq 2b$, $\phi_{T_{p-1}}(c+1) \leq b/2$, and by the inductive assumption $\phi_{T_p}(c+1) \leq b/4$, we will show that $\phi_{T_p}(c) \leq b$. Let $s_t(c) := \phi_t(c) - \phi_t(c+1)$; intuitively, $s_t(c)$ can be seen as total the number of tokens in the system which are "stacked" on their respective nodes at heights between $cd^+ + 1$ and $(c+1)d^+$. For $t \in [T_{p-1}; T_p]$, $s_t(c)$ satisfies the following bound:

$$s_t(c) = \phi_t(c) - \phi_t(c+1) \geq \phi_t(c) - \phi_{T_{p-1}}(c+1) \geq \phi_t(c) - \tfrac{b}{2}. \tag{17}$$

For any such time moment t consider the set of nodes with load at least cd^+ at time t. Within the time interval $[t+1; t+\widehat{T}]$, where the period of time $\widehat{T} = O\left(\frac{d \log n}{\mu \max\{(2(c-c_0)d^+ + 1), d^+/2 + 1\}}\right)$ follows from Lemma 3.2,

every node with a load of more than cd^+ at time t, will decrease its load below $\bar{x} + \delta d^+ + 2d^\circ + \frac{1}{2} + \frac{1}{2}\max\{(2(c-c_0)d^+), d^+/2\} \leq c_0 d^+ - d^+/2 + \frac{1}{2} + \max\{(d^+(c-c_0)), d^+/4\} \leq cd^+ + \frac{1}{2}$ (and so also below cd^+) at some moment of time during the considered time interval. By Lemma 4.2 a potential drop occurs for $\phi(c)$ in the considered interval $[t+1; t+\widehat{T}]$. More precisely, every node u with $x_t(u) \in [cd^+, cd^+ + s]$ contributes $x_t(u) - cd^+$ to both the potential drop and the value of $s_t(c)$, whereas every node u with $x_t(u) > cd^+ + s$ contributes exactly s to the potential drop and at most d^+ to the value of $s_t(c)$. Hence, we obtain from Lemma 4.2 (and the fact that $s \leq d^+$):

$$\phi_{t+\widehat{T}}(c) \leq \phi_t(c) - \frac{s}{d^+} \cdot s_t(c). \tag{18}$$

Combining (17) and (18), we obtain for any time moment $t \in [T_{p-1}, T_p]$:

$$\phi_{t+\widehat{T}}(c) \leq \phi_t(c) - \frac{s}{d^+} \cdot (\phi_t(c) - \tfrac{b}{2}). \tag{19}$$

We can transform this expression to the following form:

$$\phi_{t+\widehat{T}}(c) - \tfrac{b}{2} \leq \phi_t(c) - \tfrac{b}{2} - \frac{s}{d^+} \cdot (\phi_t(c) - \tfrac{b}{2}) = \left(1 - \frac{s}{d^+}\right) \cdot (\phi_t(c) - \tfrac{b}{2}).$$

Observe that we can fix t_p satisfying (16) so that $t_p \geq \frac{d^+}{s}\widehat{T}$ (which implies $T_p \geq T_{p-1} + \frac{d^+}{s}\widehat{T}$) where we took into account that $2(c-c_0) \geq 2(c_1 - c_0) - p$ for $c \geq c_1 - p/2$. Now, taking advantage of the monotonicity of potentials, we have from (19):

$$\phi_{T_p} \leq \phi_{(T_{p-1} + \frac{d^+}{s}\widehat{T})}(c) \leq \tfrac{b}{2} + \left(1 - \frac{s}{d^+}\right)^{\frac{d^+}{s}} \cdot (\phi_{T_{p-1}}(c) - \tfrac{b}{2}) \leq \tfrac{b}{2} + \tfrac{1}{2}(2b - \tfrac{b}{2}) = \tfrac{3}{4}b \leq b,$$

which completes the inductive proof of the bound on t_p for $c \geq c_1 - p/2$.

Moreover, for $c < c_1 - p/2$, (14) holds because $4^{(c_1-c)}2^{-p} > 1$, and $\phi_{T_p}(c) \leq (c_1 - c_0)d^+ n$ holds by the definition of potentials.

Now, taking into account (15) and (16), we can bound the time of termination of phase p_f of the process as follows:

$$T_{p_f} = T + \sum_{p=1}^{p_f} t_{p_f} = T + \sum_{p=1}^{p_f} O\left(\frac{d^+}{s} \frac{d \log n}{\mu \cdot \max\{(2(c_1 - c_0) - p)d^+ + 1, d^+/2 + 1\}}\right) =$$

$$= O\left(T + \frac{d}{s}\frac{\log n}{\mu}\left(\left(\sum_{p=1}^{2(c_1-c_0)-1} \frac{1}{2(c_1 - c_0) - p + 1/d^+}\right) + \frac{d^+}{d^+/2+1} \cdot \log\left((c_1 - c_0)d^+ n\right)\right)\right) = O\left(T + \frac{d}{s}\frac{\log^2 n}{\mu}\right),$$

where we recall that p_f was given by expression (15), and that $c_1 d^+ - c_0 d^+ = O\left(\frac{d \log n}{\mu}\right)$.

In this way, we have shown that a $(\delta + 1/2)d^+ + 2d^\circ$-unbalance is achieved in time $O\left(T + \frac{d}{s}\frac{\log^2 n}{\mu}\right)$. By using the same technics and Lemma 4.4 instead of Lemma 4.2, we can show that no node has a load of less than $\bar{x} - (\delta + 1/2)d^+ + 2d^\circ$. This gives the desired discrepancy bound of $(2\delta + 1)d^+ + 4d^\circ$. □

5 Lower Bounds

We start by showing that the cumulative fairness bounds we introduce cannot be completely discarded when improving upon the discrepancy gaps from [14]. We say an algorithm is *round-fair* (in the sense of [14]) if the load which any node u sends over its edges is either $\left\lfloor \frac{x_t(u)}{d} \right\rfloor$ or $\left\lceil \frac{x_t(u)}{d} \right\rceil$. In the following we see that if cumulative fairness is excessively ignored, then there are round-fair algorithms which have a discrepancy of at least $(c \operatorname{diam}(G) \cdot d)$ for some constant c.

Theorem 5.1. *Let G be a d-regular graph. There exists an initial distribution of tokens and a round-fair algorithm A, such that A cannot achieve a discrepancy better than $(c \operatorname{diam}(G) \cdot d)$, for some absolute constant $c > 0$.*

Proof. We will describe an initial state, corresponding to a steady state distribution, such that the flow of load along each edge e is the same in every moment of time ($f_0(e) = f_1(e) = f_2(e) = \ldots$), and the load of nodes does not change in time (however, the load of two distant nodes in the graph will be sufficiently far apart). We take two vertices u and w such that distance between them is $\mathrm{diam}(G)$. We assign to every node $v \in V$ a value $b(v)$ being the shortest path distance from v to u ($b(u) = 0$, $b(v) = 1$ for direct neighbours of u, etc.). For any given edge (v_1, v_2), we assign

$$f_0(v_1, v_2) = \min(b(v_1), b(v_2))$$

We observe, that for each v:

$$\max_{e_1, e_2 \in E_v} |f_0(e_1) - f_0(e_2)| \leq 1$$

and for each edge (v_1, v_2):

$$f_0(v_1, v_2) = f_0(v_2, v_1).$$

Thus, at each step there exists a way to assign values of $\lceil f(v) \rceil$ and $\lfloor f(v) \rfloor$ so as to achieve desired values over edges, and that the system is in the steady state. The sought value of discrepancy is achieved for the considered pair of nodes u and w whose distance in G is $\mathrm{diam}(G)$. □

The following bound shows that the stateless algorithms we design are asymptotically the best possible in terms of eventual discrepancy. Namely, any stateless algorithm is not able to achieve a discrepancy better than cd, for some constant c. This also means that the bound on the discrepancy, presented in Theorem 2.6, cannot be improved in general for the class of strictly fair algorithms.

Theorem 5.2. *Let A be an arbitrary deterministic and stateless algorithm. For every even n, there exists a d-regular graph and an initial load distribution such that A cannot achieve discrepancy better than cd, for some absolute constant $c > 0$.*

Proof. We take an arbitrary graph G with n vertices which contains a $\lfloor d/2 \rfloor$-clique C. We can construct such a graph by taking nodes numbered from 0 to $n-1$ and connecting each pair of nodes i and j with an edge if and only if $(i-j) \bmod n \in \{n - \lfloor d/2 \rfloor, \ldots, n-1, 0, 1, \ldots, \lfloor d/2 \rfloor\}$. If d is odd, we also add edges (i, j) for all $(i-j) \bmod n = n/2$. W.l.o.g. we can assume that $C = \{0, 1, \ldots, \lfloor d/2 \rfloor - 1\}$.

Let A be a deterministic and memoryless algorithm. Since A is deterministic and stateless, for each node u, at round t the load which A sends to neighbours and the load it keeps through the remainder vector depend solely on the current load of u. Let us fix $\ell = |C| - 1$. Initially, the load is distributed in such a way that every node in C has ℓ load and every other node has load 0. For any given node whose current load is ℓ, let p° denote the number of tokens kept by A at the considered node, and let $p_1, p_2, \ldots, p_d$ be the number of tokens sent by A along respective outgoing edges. Clearly, $\ell = p^\circ + \sum_{i=1}^{d} p_i$. At most ℓ of those values are positive, so we can assume w.l.o.g. that $p_d = p_{d-1} = \ldots = p_{\ell+1} = 0$.

We complete the construction in such a way that at each time step the load over every node is preserved. Let us fix $i \in C$. We design an adversary which has control over which values from $\{p_1, \ldots, p_d\}$ are sent along edges of the clique, and which chooses to send along edges of the clique the possibly nonzero values $p_1, p_2, \ldots, p_\ell$. These values will be assigned to the edges $(i, (i+1) \bmod d), (i, (i+2) \bmod d), \ldots, (i, (i-1) \bmod d)$, respectively. We assign all other values arbitrarily since they are all equal to 0. Thus, we observe that at each step loads of nodes are preserved, since the new load of all nodes having load ℓ at the end of a step is $p^\circ + \sum_{i=1}^{\ell} p_i = \ell$ in the next step. All other nodes in $V \setminus C$ will not receive any tokens and they will remain with load 0. Thus, the load of nodes in the graph does not change over rounds and the load difference of nodes in C and the nodes in $V \setminus C$ is cd, for some constant $c > 0$. □

Our final lower bounds concern variants of the Rotor-Router. The next theorem shows that for a graph without self-loops (i.e., $G = G^+$) the best possible discrepancy of the Rotor-Router is at least $c \cdot d \cdot \varphi'(G)$, where $\varphi'(G)$ is the odd girth of graph G, i.e., the length of the shortest odd length cycle over all nodes of G. This gives for an odd-length cycle a discrepancy of at least $c \cdot d \cdot \mathrm{diam}$ for some constant c.

Theorem 5.3. *Let G be any d-regular and non-bipartite graph, and let $d^+ = d$. Then, there exists an initial configuration of load on nodes such that* Rotor-Router *cannot achieve discrepancy better than $(c \cdot d\varphi(G))$, for some absolute constant $c > 0$, where $(2\varphi(G) + 1)$ is the odd girth of G.*

Proof. Let u be an arbitrary vertex belonging to the shortest odd cycle. We assign to every node $v \in V$ a value $b(v)$ being the shortest path distance from v to u ($b(u) = 0$, $b(v) = 1$ for direct neighbours of u, etc.). Observe that for any edge (v_1, v_2), we have $b(v_1) - b(v_2) \in \{-1, 0, 1\}$. Moreover, we have $b(v_1) = b(v_2)$ only if $b(v_1) \geq \varphi(G)$. Suppose $b(v_1) < \varphi(G)$. We can construct an odd length cycle $u \rightsquigarrow v_1 \to v_2 \rightsquigarrow u$ of at most $2 \cdot \varphi(G) - 1$, a contradiction.

For the construction, fix a sufficiently large integer $L > 0$ which will be linked to the average load in the system (it has no effect over the final discrepancy, we need L to be large enough to have nonnegative values of load). We will design a configuration of load in the system which will alternate between two different states, identical for all configurations in odd time steps and even time steps, respectively (thus, $f_0(e) = f_2(e) = \ldots$ and $f_1(e) = f_3(e) = \ldots$). We will describe a configuration at any time step by providing values distributed over every edge. The load distributed over edge (v_1, v_2) will only depend on the values of $b(v_1)$ and $b(v_2)$. If $b(v_1) \geq \varphi(G)$ or $b(v_2) \geq \varphi(G)$, we set $f_0(v_1, v_2) = L$, otherwise:

$$f_0(v_1, v_2) = \begin{cases} L + (\varphi(G) - \min(b(v_1), b(v_2))) & \text{if } 2|b(v_1) \text{ and } 2 \not| b(v_2), \\ L - (\varphi(G) - \min(b(v_1), b(v_2))) & \text{if } 2 \not| b(v_1) \text{ and } 2|b(v_2). \end{cases}$$

We also set:

$$f_1(v_1, v_2) = f_0(v_2, v_1). \tag{20}$$

Setting all of $f_0(e)$ and $f_1(e)$ is enough to describe every value over every edge. We now prove that such a configuration is possible for some execution of the Rotor-Router algorithm, i.e., that there exists an ordering of the outgoing edges of each node in the cycle of the Rotor-Router which leads to such alternating configurations. We observe that $f_t(v_1, v_2) + f_t(v_2, v_1) = 2L$ for $t \in \mathbb{N}$. Thus, for $t \in \mathbb{N}$ we have

$$f_t(v_1, v_2) + f_{t+1}(v_1, v_2) = 2L. \tag{21}$$

We observe, that for any node v and its two neighbours v_1, v_2, it holds

$$|f_t(v, v_1) - f_t(v, v_2)| \leq 1$$

Also, due to (21):

$$\ldots = f_t(v, v_1) - f_t(v, v_2) = -\left(f_{t+1}(v, v_1) - f_{t+1}(v, v_2)\right) = f_{t+2}(v, v_1) - f_{t+2}(v, v_2) = \ldots$$

By (20), the incoming and outgoing flows of load through edges are preserved, and because the difference between outgoing flows is alternating in signs (for two incident outgoing edges), it is always possible to choose an edge ordering in the cycle of the Rotor-Router representing such a situation. Indeed, we observe that for particular vertex v, outgoing directed edges of v can be partitioned into two sets $P_1 \cup P_2$, where in even steps edges from P_1 are given one more token than edges from P_2, and in odd steps edges from P_1 are given one less token than edges from P_2. So it is enough to select an ordering of edges for the Rotor-Router such that every edge from P_1 precedes every edge from P_2 set.

We observe that the node u alternates between loads $(L + \varphi(G)) \cdot d$ and $(L - \varphi(G)) \cdot d$, while average load of a node in this setting is exactly $L \cdot d$, which gives us the claimed discrepancy. □

As we will see below, even with $d^\circ > 0$ the Rotor-Router cannot achieve a discrepancy smaller than cd for some constant c and an arbitrary G.

Theorem 5.4. *Let G be any d-regular and non-bipartite graph, and let $d^+ > d$. Then, there exists an initial configuration of load on nodes such that* Rotor-Router *cannot achieve discrepancy smaller than $(c \cdot d)$, for some absolute constant $c > 0$.*

Proof. We follow along the lines of the previous proof, constructing a initial configuration that will alternate between two different settings. Let us pick an arbitrary vertex u. We set $f_0(e) = 2$ for all outgoing edges of u, $f_0(e) = 0$ for all incoming edges of u and $f_0(e) = 1$ for every other edge, and for all self-loops.

We observe that for every vertex the outgoing values differ by at most 1, thus making it possible to find a cycle of outgoing edges for each node, achieving such a setting. We also observe that every edge that transferred a load of 0 in one timestep will transfer and a load of 2 in following time step and vice-versa (edges which transfer a load of 1 will continue doing so in subsequent steps). Thus, vertex u alternates between loads $d^+ + d$ and $d^+ - d$, while the average load of all nodes in the graph is d^+, giving the desired load discrepancy. □

As we will see in the following, even using algorithm ROTOR-ROUTER* one cannot achieve a discrepancy less than cd, for some constant $c > 0$.

Theorem 5.5. *Let G be a d-regular and non-bipartite graph. Then there exists a start configuration such that* ROTOR-ROUTER* *cannot achieve discrepancy better than (cd), for some absolute constant $c > 0$.*

Proof. We follow the lines of the previous proofs. Every edge will have assigned a value from the set $\{0, 1\}$. The system will again alternate over two states ($f_0(e) = f_2(e) = \ldots$ and $f_1(e) = f_3(e) = \ldots$) for every edge except all of the self loops, which will satisfy: $f_0(v, v) = f_1(v, v) = \ldots = 1$. We assign values $\{0, 1\}$ arbitrarily, making sure that $\exists_u \forall_{v \in E_u} f_0(v, u) = 1$, and that $\forall_{(v_1,v_2)\in E} f_0(v_1, v_2) = 1 - f_0(v_2, v_1)$.

We proceed by ordering the outgoing edges in the rotor-router cycle at every vertex by non-increasing values of flow over those edges at $t = 0$ (more precisely, we require that the rotor moves to edges with flow value 1 before those with value 0). Having this local edge ordering, ROTOR-ROUTER* will maintain the property (for all outgoing edges):

$$f_t(v_1, v_2) + f_{t+1}(v_1, v_2) = 1.$$

We observe that u alternates between loads $d + 1$ and 1, while the average load for a node in the graph is $d/2 + 1$, thus giving the desired discrepancy. □

6 Conclusions

We have provided characterizations of conditions on deterministic algorithms for discrete load balancing, which lead to improved bounds on discrepancy after time comparable to the diffusion time T of the continuous diffusion process on the same input. It is particularly interesting to see that stateless algorithms, which are particularly robust, can give strong guarantees on the quality of the achieved balancing. We show that such algorithms can obtain a bound of $O(d)$ on the discrepancy, which is tight for these processes, after time only slightly longer than T (Table 1.1). Reducing this time to $O(T)$ constitutes an interesting and important open problem. For the broader class of cumulatively fair algorithms, we achieve a discrepancy of $O\left(d \min\left\{\sqrt{\frac{\log n}{\mu}}, \sqrt{n}\right\}\right)$. Whereas this result may admit potential improvement — we know of no cumulatively fair algorithms achieving this bound — such an improvement would require the development of completely new insights, and probably new techniques concerning the change of probability distributions of random walks in graphs in time.

The algorithms defined in our framework can be applied for the case of non-regular graphs. In this case, it is easiest to regularize the graph by adding self-loops to each node in G^+, evening out the degree of all nodes, given a known upper bound on the maximum degree in the graph. Load balancing can also be performed without such regularization, at the cost of an additional time factor equal to the ratio between the maximum and minimum degree. In this case, the outcome load distribution is proportional to the degrees of nodes, and not uniform.

Our framework also extends to the case of tasks of non-uniform length, which cannot be modelled by unit tokens. In this case, our framework of δ-fairness in cumulatively and selfishly fair algorithms easily accommodates the case of tasks of integer size, bounded by δ. For example, cumulatively fair algorithms which may be considered in this case include greedy variants of the rotor-router, in which successive tasks are sent out along the edge for the which the cumulative load so far is the smallest.

References

[1] C. Adolphs and P. Berenbrink. Distributed selfish load balancing with weights and speeds. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, PODC '12, pages 135–144, New York, NY, USA, 2012. ACM.

[2] C. Adolphs and P. Berenbrink. Improved bounds on diffusion load balancing. In *IPDPS*, 2012.

[3] H. Akbari and P. Berenbrink. Parallel rotor walks on finite graphs and applications in discrete load balancing. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '13, pages 186–195, New York, NY, USA, 2013. ACM.

[4] H. Akbari, P. Berenbrink, and T. Sauerwald. A simple approach for adapting continuous load balancing processes to discrete settings. In *PODC*, pages 271–280, 2012.

[5] P. Berenbrink, C. Cooper, T. Friedetzky, T. Friedrich, and T. Sauerwald. Randomized diffusion for indivisible loads. In *22nd*, pages 429–439. SIAM, 2011.

[6] J. Cooper, B. Doerr, T. Friedrich, and J. Spencer. Deterministic random walks on regular trees. *Random Struct. Algorithms*, 37(3):353–366, 2010.

[7] B. Doerr and T. Friedrich. Deterministic random walks on the two-dimensional grid. *Combinatorics, Probability and Computing*, 18(1-2):123–144, 2009.

[8] T. Friedrich, M. Gairing, and T. Sauerwald. Quasirandom load balancing. In *SODA*, pages 1620–1629. SIAM, 2010.

[9] T. Friedrich and T. Sauerwald. Near-perfect load balancing by randomized rounding. In *STOC*, pages 121–130, 2009.

[10] T. Friedrich and T. Sauerwald. The cover time of deterministic random walks. In *COCOON*, pages 130–139, 2010.

[11] S. Kijima, K. Koga, and K. Makino. Deterministic random walks on finite graphs. In *ANALCO*, pages 16–25, 2012.

[12] D. Levin, Y. Peres, and E. Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2006.

[13] V. Priezzhev, D. Dhar, A. Dhar, and S. Krishnamurthy. Eulerian walkers as a model of self-organized criticality. *Phys. Rev. Lett.*, 77:5079–5082, Dec 1996.

[14] Y. Rabani, A. Sinclair, and R. Wanka. Local divergence of Markov chains and the analysis of iterative load-balancing schemes. In *FOCS*, pages 694–703, nov 1998.

[15] T. Sauerwald and H. Sun. Tight bounds for randomized load balancing on arbitrary network topologies. In *FOCS*, pages 341–350, 2012. A revised and extended version is available online at: http://arxiv.org/abs/1201.2715.

[16] R. Subramanian and I. Scherson. An analysis of diffusive load-balancing. In *ACM Symposium on Parallel Algorithms and Architectures*, SPAA '94, pages 220–225, New York, NY, USA, 1994. ACM.

[17] T. Shiraga, Y. Yamauchi, S. Kijimam, and M. Yamashita. Deterministic Random Walks for Rapidly Mixing Chains. `arXiv:1311.3749`, 2013.

APPENDIX

Proof of Proposition 2.3

Proof. The reformulation of algorithm A as algorithm A' proceeds as follows. For all edges $e \in E(G)$ (i.e., except for self-loops), in every step A' places the same amount of load on e as A. However, in A' load may be retained on nodes in a different way, being placed in the remainder $r_t(u)$ rather than on self-loops at node $u \in V$. To prove, that it is always possible, we proceed by induction.

Specifically, at a fixed moment of time t, let $f_t(e)$ be the amount of load put on an edge e by algorithm A, and $f'_t(e)$ be the amount of load put on an edge by A', and let $F_t(e)$ and $F'_t(e)$ be the respective cumulative loads for algorithms A and A'. Algorithm A' processes all edges (including self-loops) sequentially and verifies if sending this amount of load along e would satisfy the cumulative fairness condition up to time t with respect to all edges outgoing from u already processed.

Let e_1 be the edge or self-loop that violates the cumulative load property for A', that is there exists an incident edge or self-loop e_2 such that $|(F'_{t-1}(e_1)+f_t(e_1))-(F'_{t-1}(e_2)+f_t(e_2))| > \delta$. Since $|f_t(e_1)-f_t(e_2)| \leq 1$, and $|F'_{t-1}(e_1) - F'_{t-1}(e_2)| \leq \delta$ (from inductive assumption), we get that

$$(F'_{t-1}(e_1) + f_t(e_1)) - (F'_{t-1}(e_2) + f_t(e_2)) \in \{\delta + 1, -\delta - 1\} \tag{22}$$

(without loss of generality we can assume that this value is $\delta + 1$). Moreover, we can show that for every e'_2 such that the pair e_1, e'_2 violates cumulative fairness, the value (22) is $\delta+1$ (otherwise $F'_{t-1}(e_1)-F'_{t-1}(e_2) = \delta$ and $F'_{t-1}(e_1) - F'_{t-1}(e'_2) = -\delta$ imply $F'_{t-1}(e'_2) - F'_{t-1}(e_2) = 2\delta$ which contradicts the inductive assumption). We can also observe that e_1 is a loop (attached to vertex u), since non-loop edges satisfy cumulative fairness for A. Thus it is enough to set $f'_t(e_1) = f_t(e_1) - 1$ and increase $r_t(u)$ by one (in the mirror scenario with a value of $-\delta - 1$ in (22) we would set $f'_t(e_1) = f_t(e_1) + 1$ and decrease $r_t(u)$ by one). It is easy to observe that this makes e_1 satisfy δ-fairness with every other edge incident to u. After processing all edges and self-loops and edges in this way, we eventually obtain that cumulative fairness is preserved, and moreover $|r'_t(u)| \leq d^\circ \leq d^+$. □

Proof of Lemma 4.3

Proof. The proof is similar to Lemma 4.1. Fix $c \in \mathbb{N}$. At any time t, we will divide the set L of tokens circulating in the system into two groups: the set of *black* tokens L_t^- and the set of *red* tokens L_t^+, with $L = L_t^- \cup L_t^+$. Colors of tokens persist over time unless they are explicitly recolored. For $t = 1$, for each node u we color exactly $|L_1^-(u)| = \min\{x_1(u), cd^+ + s\}$ tokens at u black, and the remaining tokens at u red. In every time step, we follow two rules concerning token distribution:

(1) The number of black tokens leaving u along outgoing edges is at most c.

(2) At the start of each subsequent step t, we recolor some red tokens to black, so that the total number of black tokens located at a node u is exactly $|L_t^-(u)| = \min\{x_t(u), cd^+ + s\}$.

We note that both rules of token circulation are well defined. The proof proceeds by induction. To prove the correctness of rule (1) at step t, observe that, by the definition of strictly fair algorithms, for any node u we either have $x_t(u) \leq cd^+$ and then node u sends at most c black tokens along each of its edges and self-loops, or $x_t(u) > cd^+$, and then node u sends over $\max\{\min\{x_t(u) - cd^+, s\}, 0\}$ many self-loops $c + 1$ black tokens and c along all other edges. In the first case, rule (1) is correct regardless of how u distributes tokens of different colors; in the second case, let $s' = \max\{\min\{x_t(u) - cd^+, s\}, 0\}$. Node u has exactly $cd^+ + s'$ black tokens, and we can require that it sends exactly $c + 1$ of its black tokens along s' many arbitrary self-loops, since the algorithm is s-selfish, and exactly c along each of its other edges. To prove the correctness of rule (2), we note that by the correctness of rule (1) for the preceding time step, the number of black tokens arriving at u along edges and self-loops can be upper-bounded by $\min\{x_t(u), cd^+ + s\}$. Hence, no recoloring of tokens from black to red is ever required.

We now observe that the potential $\phi'_t(c)$ is by definition the number of missing black tokens such that every node has $cd^+ + s$ of them. Indeed, we have:

$$\phi'_t(c) = \sum_{u \in V} (cd^+ \ + s - \min\{x_t(u), cd^+ + s\}) = (cd^+ + s) \cdot n - \sum_{u \in V} |L_t^-(u)| = (cd^+ + s) \cdot n - |L_t^-|.$$

The monotonicity condition (i) for the potential follows immediately from the fact that no new red tokens appear in the system. To prove (ii), we will show that the number of tokens being recolored from red to black in time step t is at least $\Delta'_t(c, u)$. Note, that a red token, which is recolored in black, will decrease the potential by 1.
Indeed, suppose that at time $t-1$ we had for a node u: $x_{t-1}(u) = cd^+ + s - i$, for an integer $i \geq 1$. Then, by the definition of the selfishness of algorithm A, at least $i' = \min\{i, s\}$ self-loops carry at most c tokens in step t. Intuitively, each of them can 'trap' a red token. By rule (1) of the token circulation process, every neighbour of u sent at most c black tokens. Thus, the number of black tokens arriving at u at time t is at most $cd^+ + s - i'$, and the number of red tokens which u receives is at least $\max\{x_t(u) - (cd^+ + s - i'), 0\}$. Therefore, for $x_{t-1}(u) < cd^+ + s$, since at most i' self-loops 'trap' a red token we have that the number of red tokens which are repainted black at node u at time t is at least (by rule (2) of recoloring)

$$\begin{aligned}
&\min\{\max\{x_t(u) - (cd^+ + s - i'), 0\}, i'\} \\
&= \min\{\max\{\ x_t(u) - (cd^+ - \min\{cd^+ - x_{t-1}(u), 0\}), 0\}, \min\{cd^+ + s - x_{t-1}(u), s\}\} \\
&= \max\{\min\{x_t(u) - x_{t-1}(u), s, x_t(u) - cd^+, cd^+ + s - x_{t-1}(u)\}, 0\} = \Delta'_t(c, u),
\end{aligned}$$

and for $x_{t-1}(u) \geq cd^+ + s$ we have $\Delta'_t(c, u) = 0$, which completes the proof of (ii).

□