Finite-Time Distributed Flow Balancing

Christoforos N. Hadjicostis, *Fellow, IEEE*, Alejandro D. Domínguez-García, *Senior Member, IEEE*, and Apostolos I. Rikos, *Member, IEEE*

***Abstract*—We consider a flow network that is described by a digraph, each edge of which can admit a flow within a certain interval, with nonnegative end points that correspond to lower and upper flow limits. We propose and analyze a distributed iterative algorithm for solving, in *finite time*, the so-called feasible circulation problem, which consists of computing flows that are *admissible* (i.e., within the given intervals at each edge) and *balanced* (i.e., the total in-flow equals the total out-flow at each node). The algorithm assumes a communication topology that allows bidirectional message exchanges between pairs of nodes that are physically connected (i.e., nodes that share a directed edge in the physical topology) and is shown to converge to a feasible and balanced solution as long as the necessary and sufficient circulation conditions are satisfied with strict inequality. In case the initial flows and flow limits are commensurable (i.e., they are integer multiples of a given constant), then the proposed algorithm reduces to a previously proposed finite-time balancing algorithm, for which we provide an explicit bound on the number of steps required for termination.**

***Index Terms*—Flow networks, Balancing, Finite-time, Feasible circulation, Distributed algorithms, Distributed balancing**

I. Introduction

We consider a flow network comprised of multiple nodes that are interconnected via some directed links through which a certain commodity can flow; we refer to this network as the physical digraph or topology. We assume that the flow on each link is constrained to lie within an interval, with nonnegative end points that correspond to link lower and upper capacity limits. The objective is to find an admissible and balanced flow assignment, i.e., find flows on all the links that are within the corresponding capacity limits, such that the sum of the in-flows is equal to the sum of the out-flows at each node. In the network flow literature, this problem is referred to as the feasible circulation problem (see, e.g., [1], [2]). The feasible circulation problem has been studied quite extensively with most algorithmic approaches implicitly assuming the existence of a centralized processor with access to all data defining the problem, although there is some recent work that has focused on developing distributed algorithmic approaches (see Chapter 5 of [3] and [4], as well as the review of relevant literature at the end of this section). The feasible circulation problem is a special case of the feasible distribution problem (see, e.g., [2]), which is closely related to the max-flow min-cut problem (see, for example, [5], [6], [7]).

A survey of many important applications of the feasible circulation/distribution problems, as well as related optimization problems, can be found in [8]. In particular, such problems arise in the operation of various infrastructure networks (such as water, power, traffic or communication networks), where we need to match demand and supply in real-time [9], [10], [11]. The feasible circulation problem can also be viewed as the problem of weight balancing a given digraph, which finds numerous applications in distributed adaptive control or synchronization in complex networks, [12], [13]. For these reasons, distributed algorithms for weight balancing is a topic that has attracted attention recently [14], [15], [16], [17], [18].

In this paper, we propose a finite-time distributed algorithm (Algorithm 2) that allows the nodes of a flow network to compute a solution to the feasible circulation problem for the case when the conditions for the existence of a solution are satisfied with strict inequality. The proposed algorithm operates in an iterative fashion and assumes that nodes that are physically connected (i.e., nodes that are connected via a physical edge in the flow network) can communicate in a bidirectional manner. Unlike existing distributed algorithms which produce balanced flows asymptotically when operating on real-valued flows (see, for example, [18], [19]), the algorithm in this paper is shown to complete in *finite time* (which is an important advantage when the computation of balanced flows is a precursor to a subsequent task) and also avoids running into finite precision problems in digital implementations, because it operates with quantized flow values.

When the initial flows and flow limits are commensurable (i.e., they are integer multiples of the same constant c), the proposed algorithm reduces to a previously proposed algorithm (Algorithm 1 in this paper, presented in [20]). Here, we provide an upper bound on the number of steps required for Algorithm 1 to terminate, which we subsequently use to establish termination of Algorithm 2. Indeed, Algorithm 2 will converge in finite time even when the initial flows and flow limits are *not* commensurable values, as long as the necessary and sufficient circulation conditions are satisfied with *strict* inequality. The main idea is to first crudely quantize the flows and flow limits to some precision (so that they are commensurable and Algorithm 1 is applicable), and subsequently increase precision for flows and flow limits as necessary (in a way that still maintains commensurability).

The work of C. N. Hadjicostis has been supported in part by the European Commission's Horizon 2020 research and innovation programme under grant agreement No 739551 (KIOS CoE). The work of A. D. Domínguez-García has been partially funded by the Advanced Research Projects Agency-Energy (ARPA-E) within the NODES program, under Award DE-AR0001189. Any opinions, findings, and conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of EC or ARPA-E.

C. N. Hadjicostis is with the Department of Electrical and Computer Engineering at the University of Cyprus, Nicosia, Cyprus, and also with the Department of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign. E-mail: chadjic@ucy.ac.cy.

Alejandro D. Domínguez-García is with the Department of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA. E-mail: aledan@illinois.edu.

Apostolos I. Rikos is with the Division of Decision and Control Systems, KTH Royal Institute of Technology, SE-100 44 Stockholm, Sweden. E-mail: rikos@kth.se.

II. Preliminaries

A digraph (directed graph) of order n ($n \geq 2$), is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} - \{(v_j, v_j) \mid v_j \in \mathcal{V}\}$ is the set of edges, where $(v_j, v_i) \in \mathcal{E}$ if there is a directed edge from node v_i to node v_j. A digraph is called *strongly connected* if for each pair of vertices, $v_j, v_i \in \mathcal{V}$, $v_j \neq v_i$, there exists a directed *path* from v_i to v_j i.e., we can find a sequence of vertices $v_i \equiv v_{l_0}, v_{l_1}, \ldots, v_{l_t} \equiv v_j$ such that $(v_{l_{\tau+1}}, v_{l_\tau}) \in \mathcal{E}$ for $\tau = 0, 1, \ldots, t-1$. All nodes from which node v_j can be reached via a directed edge are said to be in-neighbors of node v_j and belong to the set $\mathcal{N}_j^- = \{v_i \in \mathcal{V} \mid (v_j, v_i) \in \mathcal{E}\}$. The cardinality of $\mathcal{N}_j^-$ is called the *in-degree* of v_j and is denoted by $\mathcal{D}_j^- = |\mathcal{N}_j^-|$. The nodes that can be reached from node v_j via a directed edge are said to be its out-neigbors and belong to the set $\mathcal{N}_j^+ = \{v_l \in \mathcal{V} \mid (v_l, v_j) \in \mathcal{E}\}$. The cardinality of $\mathcal{N}_j^+$ is called the *out-degree* of v_j and is denoted by $\mathcal{D}_j^+ = |\mathcal{N}_j^+|$.

A distributed system whose components can exchange a certain commodity via directed links can conveniently be captured by a digraph $\mathcal{G}_p = (\mathcal{V}, \mathcal{E}_p)$, which we will refer to as the *flow topology* or *physical digraph/topology*. Given $\mathcal{G}_p$, we can associate nonnegative flows $f_{ji} \in \mathbb{R}$ on each edge $(v_j, v_i) \in \mathcal{E}_p$. In this paper, the flow f_{ji} will be restricted to lie in a real interval $[l_{ji}, u_{ji}]$, where $0 \leq l_{ji} \leq f_{ji} \leq u_{ji}$. We can then define the flow, lower limit, and upper limit $(n \times n)$-dimensional matrices, denoted as F, L and U, respectively, as follows: $F(j,i) = f_{ji}$, $L(j,i) = l_{ji}$, and $U(j,i) = u_{ji}$ (and $f_{ji} = l_{ji} = u_{ji} = 0$ when $(v_j, v_i) \notin \mathcal{E}_p$). Following the notation introduced previously, we will denote the *physical* in-neighbors of node v_j by $\mathcal{N}_j^- = \{v_i \in \mathcal{V} \mid (v_j, v_i) \in \mathcal{E}_p\}$, and the *physical* in-degree of v_j by $\mathcal{D}_j^-$. Similarly, we will denote the *physical* out-neighbors of node v_j by $\mathcal{N}_j^+ = \{v_l \in \mathcal{V} \mid (v_l, v_j) \in \mathcal{E}_p\}$, and the *physical* out-degree of v_j by $\mathcal{D}_j^+$. We will also use $\mathcal{D}_j = \mathcal{D}_j^- + \mathcal{D}_j^+$ to denote the *total degree* of node v_j.

For the purposes of developing our distributed flow balancing algorithms, we will rely on a communication topology that is captured by a digraph $\mathcal{G}_c = (\mathcal{V}, \mathcal{E}_c)$ that includes bidirectional edges between all nodes that are neighbors in the physical topology. In other words,

$$\mathcal{E}_c = \{(v_j, v_i), (v_i, v_j) \mid (v_j, v_i) \in \mathcal{E}_p\} \ . \tag{1}$$

This implies that all nodes that are neighbors in the physical topology can communicate in a bidirectional manner.

Given a physical digraph $\mathcal{G}_p = (\mathcal{V}, \mathcal{E}_p)$ of order n along with a flow assignment $F \in \mathbb{R}^{n \times n}$ with $F(j,i) = f_{ji}$, we define the following quantities.

Definition 1: The total *in-flow* of node v_j is denoted by f_j^-, and is defined as $f_j^- = \sum_{v_i \in \mathcal{N}_j^-} f_{ji}$, whereas the total *out-flow* of node v_j is denoted by f_j^+, and is defined as $f_j^+ = \sum_{v_l \in \mathcal{N}_j^+} f_{lj}$. The *flow balance* or *balance* of node v_j is denoted by b_j and is defined as $b_j = f_j^- - f_j^+$.

Definition 2: The *total imbalance* (or *absolute balance*) of digraph $\mathcal{G}_p$ is denoted by ε and is defined as $\varepsilon = \sum_{j=1}^n |b_j|$.

Definition 3: A digraph $\mathcal{G}_p = (\mathcal{V}, \mathcal{E}_p)$ of order n, along with a flow assignment F, is called flow-balanced if its *total imbalance* (or *absolute balance*) is 0, i.e., $\varepsilon = \sum_{j=1}^n |b_j| = 0$.

The distributed algorithms we will develop are iterative, and we will use k to index the iterations. Thus, we will use $f_{ji}[k]$ to denote the estimate, at iteration k, of the flow value on the link $(v_j, v_i) \in \mathcal{E}_p$; $f_j^+[k]$ to denote the estimate, at iteration k, of the total out-flow of node v_j, and so forth.

Flow Assignment Problem: We are given a physical (strongly connected) digraph $\mathcal{G}_p = (\mathcal{V}, \mathcal{E}_p)$, and a communication digraph $\mathcal{G}_c = (\mathcal{V}, \mathcal{E}_c)$ (with $\mathcal{E}_c$ as defined in (1)), as well as lower and upper bounds l_{ji} and u_{ji} ($0 \leq l_{ji} \leq u_{ji}$) on each edge $(v_j, v_i) \in \mathcal{E}_p$. We want to develop a distributed iterative algorithm that respects the communication restrictions imposed by the communication digraph $\mathcal{G}_c$, and allows the nodes to iteratively adjust the flows on their outgoing edges, so that they obtain a set of flows $\{f_{ji} \mid (v_j, v_i) \in \mathcal{E}_p\}$ that satisfy the following conditions:

$C1$. $0 \leq l_{ji} \leq f_{ji} \leq u_{ji}$ for each edge $(v_j, v_i) \in \mathcal{E}_p$;
$C2$. $f_j^+ = f_j^-$ for every $v_j \in \mathcal{V}$.

Theorem 1: (Circulation Theorem [1]) Consider a strongly connected digraph $\mathcal{G}_p = (\mathcal{V}, \mathcal{E}_p)$, with lower and upper bounds l_{ji} and u_{ji} ($0 \leq l_{ji} \leq u_{ji}$) on each edge $(v_j, v_i) \in \mathcal{E}_p$. The necessary and sufficient condition for the existence of a set of flows $\{f_{ji} \mid (v_j, v_i) \in \mathcal{E}_p\}$ that satisfy $C1$ and $C2$, is the following: for each $\mathcal{S}$, $\mathcal{S} \subset \mathcal{V}$, we have

$$\sum_{(v_j, v_i) \in \mathcal{E}_\mathcal{S}^-} l_{ji} \leq \sum_{(v_l, v_j) \in \mathcal{E}_\mathcal{S}^+} u_{lj} \ , \tag{2}$$

where

$$\mathcal{E}_\mathcal{S}^- = \{(v_j, v_i) \in \mathcal{E}_p \mid v_j \in \mathcal{S},\ v_i \in \mathcal{V} - \mathcal{S}\} \ , \tag{3}$$

$$\mathcal{E}_\mathcal{S}^+ = \{(v_l, v_j) \in \mathcal{E}_p \mid v_j \in \mathcal{S},\ v_l \in \mathcal{V} - \mathcal{S}\} \ . \tag{4}$$

Assuming the above circulation conditions hold, a variety of centralized algorithms for obtaining such flows exist (see, e.g., [1], [2]). One possibility is the feasible distribution algorithm, which is a centralized algorithm that iteratively attempts to find a particular type of path between the set of positively-balanced nodes and the set of negatively-balanced nodes, using a process termed *arc discrimination* [2]. If a solution exists, this centralized algorithm computes it in finite time.

Definition 4: When the circulation conditions in (2) hold, we will say that they hold *with strict inequality* if (2) holds with strict inequality for *all* subsets $\mathcal{S}$, $\mathcal{S} \subset \mathcal{V}$.

III. Distributed Flow Assignment Algorithm

We start by reviewing the balancing algorithm in [20] that assumes that flow limits and initial flows are commensurable (i.e., they are integer multiples of some constant c).

A. Basic Version of Distributed Balancing Algorithm

Initialization. At initialization, each node is aware of the feasible flow interval on each of its incoming and outgoing edges, i.e., node v_j is aware of l_{ji}, u_{ji} for each $v_i \in \mathcal{N}_j^-$ and l_{lj}, u_{lj} for each $v_l \in \mathcal{N}_j^+$ (all of which are assumed to be commensurable values). Furthermore, the flows are initialized at the lower limit of the feasible interval, i.e., $f_{ji}[0] = l_{ji}$. [This initialization is not critical and could be any value in the feasible flow interval $[l_{ji}, u_{ji}]$ that is commensurable with

the flow limits.] Each node also assigns an order in the set $\{0, 1, 2, ..., (\mathcal{D}_j - 1)\}$ to each of its outgoing and incoming edges. It also initializes an internal index variable FTC_j (next *flow to change* by node v_j), which indicates the order of the edge whose flow will be changed next, to be $FTC_j = 0$.

Iteration. At each iteration $k \geq 0$, node v_j is aware of the flow estimates on its incoming edges, $\{f_{ji}[k] \,|\, v_i \in \mathcal{N}_j^-\}$, and outgoing edges, $\{f_{lj}[k] \mid v_l \in \mathcal{N}_j^+\}$, and updates them using the following three steps:

[Step 1.] If $b_j[k] > 0$, node v_j attempts to change the flow at one of its incoming edges, $\{f_{ji}[k+1] \mid v_i \in \mathcal{N}_j^-\}$, or one of its outgoing edges, $\{f_{lj}[k+1] \mid v_l \in \mathcal{N}_j^+\}$, in a way that drives its balance $b_j[k+1]$ to zero (at least if no other changes are inflicted on the flows). More specifically, node v_j chooses to change the flow associated with the edge that has order FTC_j. There are two cases to consider:

(i) If this edge is an incoming edge (v_j, v_i), then node v_j attempts to change the flow on this edge as $f_{ji}^{(j)}[k+1] = f_{ji}[k] - b_j[k]$; we will denote this change as $\Delta f_{ji}^{(j)} = -b_j[k]$. Note that for all other edges $(v_j, v_{i'})$, $v_{i'} \in \mathcal{N}_j^- \setminus \{v_i\}$, and (v_l, v_j), $v_l \in \mathcal{N}_j^+$, node v_j does not attempt any change, i.e., $\Delta f_{ji'}^{(j)}[k] = 0$, $v_{i'} \in \mathcal{N}_j^- \setminus \{v_i\}$, and $\Delta f_{lj}^{(j)}[k] = 0$, $v_l \in \mathcal{N}_j^+$.

(ii) If this edge is an outgoing edge (v_l, v_j), then node v_j attempts to change the flow on this edge as $f_{lj}^{(j)}[k+1] = f_{lj}[k] + b_j[k]$; we will denote this change as $\Delta f_{lj}^{(j)} = +b_j[k]$. Note that for all other edges (v_j, v_i), $v_i \in \mathcal{N}_j^-$, and $(v_{l'}, v_j)$, $v_{l'} \in \mathcal{N}_j^+ \setminus \{v_l\}$, node v_j does not attempt any change, i.e., $\Delta f_{ji}^{(j)}[k] = 0$, $v_i \in \mathcal{N}_j^-$, and $\Delta f_{l'j}^{(j)}[k] = 0$, $v_{l'} \in \mathcal{N}_j^+ \setminus \{v_l\}$.

Also, once a change is imposed to the edge associated with FTC_j, node v_j updates its index for the next flow to change to $FTC_j = (FTC_j + 1) \mod \mathcal{D}_j$. This ensures that each node considers all of its edges, in a round-robin fashion, each time it has a positive balance. [The next time node v_j needs to change the flow of an incoming/outgoing edge, it will continue from the edge it stopped the previous time.]

[Step 2.] For each edge $(v_j, v_i) \in \mathcal{E}_p$, both nodes v_j and v_i calculate the interim value of the flow as

$$\tilde{f}_{ji}[k+1] = f_{ji}[k] + \Delta f_{ji}^{(j)}[k] + \Delta f_{ji}^{(i)}[k] \,, \tag{5}$$

where $\Delta f_{ji}^{(j)}[k] \leq 0$ and $\Delta f_{ji}^{(i)}[k] \geq 0$ (for many edges $(v_j, v_i) \in \mathcal{E}_p$, we will have $\Delta f_{ji}^{(j)}[k] = 0$ or $\Delta f_{ji}^{(i)}[k] = 0$).

[Step 3.] Value $f_{ji}[k+1]$ is updated as follows (depending if $\tilde{f}_{ji}[k+1]$ lies below, within, or above the interval $[l_{ji}, u_{ji}]$):

$$f_{ji}[k+1] = \begin{cases} \tilde{f}_{ji}[k+1], & \text{if } l_{ji} \leq \tilde{f}_{ji}[k+1] \leq u_{ji} \,, \\ u_{ji}, & \text{if } \tilde{f}_{ji}[k+1] > u_{ji} \,, \\ l_{ji}, & \text{if } \tilde{f}_{ji}[k+1] < l_{ji} \,. \end{cases} \tag{6}$$

Once the values $\{f_{ji}[k+1] \mid (v_j, v_i) \in \mathcal{E}_p\}$ are obtained, the iteration steps 1–3 are repeated. The pseudocode for the algorithm is provided as Algorithm 1 and can also be found in [20], which also provides some intuition about the operation of the algorithm and examples.

Algorithm 1: Finite-Time Distributed Flow Balancing

Each node $v_j \in \mathcal{V}$ has the following input/output.

Input: l_{ji}, u_{ji}, $\forall v_i \in \mathcal{N}_j^-$ and l_{lj}, u_{lj}, $\forall v_l \in \mathcal{N}_j^+$
Output: f_{ji}, $\forall v_i \in \mathcal{N}_j^-$ and f_{lj}, $\forall v_l \in \mathcal{N}_j^+$

Each node $v_j \in \mathcal{V}$ separately does the following.

Initialization:
1. Set $f_{lj}[0] = l_{lj}$, for all $(v_l, v_j) \in \mathcal{E}_p$
2. Assign a unique order in $\{0, 1, 2, ..., (\mathcal{D}_j - 1)\}$ to each incoming and each outgoing edge
3. Set $FTC_j = 0$

Iteration: foreach *iteration,* $k = 0, 1, ...,$ **do**

Determine: $f_j^+[k]$, $f_j^-[k]$, $b_j[k]$

Set:
$\Delta f_{ji}^{(j)}[k] = 0$, $\forall v_i \in \mathcal{N}_j^-$; $\Delta f_{lj}^{(j)}[k] = 0$, $\forall v_l \in \mathcal{N}_j^+$
If $b_j[k] > 0$, then
1. Select the edge (v_j, v_i) $\big($or $(v_l, v_j)\big)$ associated with FTC_j and set
$\Delta f_{ji}^{(j)}[k] = -b_j[k]$ (or $\Delta f_{lj}^{(j)}[k] = +b_j[k]$)
2. Set $FTC_j = (FTC_j + 1) \mod \mathcal{D}_j$

Transmit:
$\Delta f_{ji}^{(j)}$ to in-neighbor $v_i \in \mathcal{N}_j^-$
$\Delta f_{lj}^{(j)}$ to out-neighbor $v_l \in \mathcal{N}_j^+$

Receive:
$\Delta f_{ji}^{(i)}$ from in-neighbor $v_i \in \mathcal{N}_j^-$
$\Delta f_{lj}^{(l)}$ from out-neighbor $v_l \in \mathcal{N}_j^+$

Set: For all $v_i \in \mathcal{N}_j^-$ and for all $v_l \in \mathcal{N}_j^+$
$\tilde{f}_{ji}[k+1] = f_{ji}[k] + \Delta f_{ji}^{(j)}[k] + \Delta f_{ji}^{(i)}[k]$
$\tilde{f}_{lj}[k+1] = f_{lj}[k] + \Delta f_{lj}^{(l)}[k] + \Delta f_{lj}^{(j)}[k]$

$$f_{ji}[k+1] = \begin{cases} \tilde{f}_{ji}[k+1], & \text{if } l_{ji} \leq \tilde{f}_{ji}[k+1] \leq u_{ji} \\ u_{ji}, & \text{if } \tilde{f}_{ji}[k+1] > u_{ji} \\ l_{ji}, & \text{if } \tilde{f}_{ji}[k+1] < l_{ji} \end{cases}$$

$$f_{lj}[k+1] = \begin{cases} \tilde{f}_{lj}[k+1], & \text{if } l_{lj} \leq \tilde{f}_{lj}[k+1] \leq u_{lj} \\ u_{lj}, & \text{if } \tilde{f}_{lj}[k+1] > u_{lj} \\ l_{lj}, & \text{if } \tilde{f}_{lj}[k+1] < l_{lj} \end{cases}$$

B. Enhanced Version of Distributed Balancing Algorithm

The enhanced version of the distributed balancing algorithm, referred to as Algorithm 2, relies on the following key observation: if one quantizes the given (not necessarily commensurable) flow limits to certain *stricter*[1] flow limit values, such that these stricter flow limit values, as well as the chosen initial flow values, are commensurable (integer multiples of some constant), then Algorithm 1 will complete in finite time, as long as these stricter quantized flow limits satisfy the circulation conditions. In fact, due to the assumption that the circulation conditions are satisfied with strict inequality, finding such stricter flow limits that are commensurable is always possible (this is explained in detail in Lemma 1 later

[1]Given lower flow limit l_{ji} (upper flow limit u_{ji}) on edge (v_j, v_i), a stricter lower flow limit $\tilde{l}_{ji}$ (stricter upper flow limit $\tilde{u}_{ji}$) is such that it satisfies $\tilde{l}_{ji} \geq l_{ji}$ ($\tilde{u}_{ji} \leq u_{ji}$).

in the paper). One big challenge, however, is the fact that the nodes need to coordinate, in a distributed manner, how to choose and, if necessary, increasingly refine the chosen stricter flow limits and initial values. Moreover, this increasing refinement needs to be done in a way that ensures that all stricter flow limits and flow values remain commensurable at all iterations.

The algorithm proposed in this section allows the stricter limits of each edge to (i) be agreed upon by the nodes in the network in a distributed manner, and (ii) be calculated based on increasingly refined uniform quantization levels, which guarantee that flows and flow limits are commensurable at all times. For the execution of the algorithm, each node needs to have knowledge of the number of edges $m = |\mathcal{E}_p|$ (or an upper bound on m) and, in order to determine whether to refine the flow limits, the nodes need to execute a max-consensus protocol [21] (a simple distributed algorithm that can be completed in finite time, of at most n steps, where n is the number of nodes).

The algorithm operates by choosing the stricter limits on each edge to be quantized values that are integer multiples of $c = \left(\frac{1}{2}\right)^{\alpha}$, where α is a nonnegative integer (initially set to zero and increased if necessary). More specifically, the lower limit on edge (v_j, v_i) is chosen to be $\tilde{l}_{ji} = L_{ji}c$ where L_{ji} is an integer that satisfies $L_{ji} = \left\lceil \frac{l_{ji}}{c} \right\rceil$, and the upper limit is chosen to be $\tilde{u}_{ji} = U_{ji}c$ where U_{ji} is an integer that satisfies $U_{ji} = \left\lfloor \frac{u_{ji}}{c} \right\rfloor$. Clearly, we have stricter limits, i.e. $\tilde{l}_{ji} \geq l_{ji}$ and $\tilde{u}_{ji} \leq u_{ji}$. Note that if $\tilde{l}_{ji}$ is greater than $\tilde{u}_{ji}$ there is no feasible assignment for flow f_{ji}, which means that we need to use a more refined quantization, i.e., increase α. For example, suppose we have $l_{ji} = 1.07$ and $u_{ji} = 1.49$; if we take $c = 1/2$ ($\alpha = 1$), we get $\tilde{l}_{ji} = 3c = 1.5$ and $\tilde{u}_{ji} = 2c = 1$ (i.e., no flow is feasible); however, with $c' = 1/4$ ($\alpha = 2$), we have $\tilde{l}_{ji} = 5c' = 1.25$ and $\tilde{u}_{ji} = 5c' = 1.25$ (i.e., the only feasible flow is $f_{ji} = 1.25$); whereas with $c'' = 1/8$ ($\alpha = 3$), we have $\tilde{l}_{ji} = 9c'' = 1.125$ and $\tilde{u}_{ji} = 11c'' = 1.375$ (i.e., the flow f_{ji} can satisfy $1.125 \leq f_{ji} \leq 1.375$).

The algorithm we present in this section allows quantization to be performed with a (possibly different) power of $1/2$ on each edge, depending on the refinement that is needed. More specifically, at any given iteration $c_{ji}[k]$ is a power of $1/2$ that captures the constant used for refining the flow limits l_{ji} and u_{ji} for edge (v_j, v_i). A pair of connected nodes v_j and v_i, agree to refine $c_{ji}[k]$ to the next higher power of $1/2$ according to the results of a max-consensus protocol.

The algorithm operates in phases, each of which runs for $4m^2$ iterations; during each phase, the lower and upper stricter flow limits remain unchanged on all edges. At the end of each phase, the nodes run a max-consensus protocol to determine whether these stricter flow limits need to be relaxed by increasing the quantization refinement. In particular, the max-consensus aims to determine whether (i) at least one node with negative balance has had its balance increase (in which case the nodes maintain the same lower and upper limits on each edge, i.e., $c_{ji}[k+1] = c_{ji}[k]$ for every edge); (ii) there is at least one node with negative balance but all nodes with negative balance have maintained the same balance throughout the phase (which is an indication that the current values of the stricter flow limits violate the circulations conditions, i.e., nodes need to refine the lower and upper limits on each edge so that $c_{ji}[k+1] = c_{ji}[k] + 1$ for every edge); (iii) there are no nodes with negative balance (in which case flow balancing has been achieved).

The basic operations of the enhanced version of the algorithm are summarized below.

Initialization. At initialization, each node is aware of the lower and upper flow limits on each of its incoming and outgoing edges, i.e., node v_j is aware of l_{ji}, u_{ji} for each $v_i \in \mathcal{N}_j^-$ and l_{lj}, u_{lj} for each $v_l \in \mathcal{N}_j^+$. It sets the quantization value $c_{ji}[0] = 1$ and calculates $L_{ji} = \lceil l_{ji}/c_{ji}[0] \rceil$ and $U_{ji} = \lfloor u_{ji}/c_{ji}[0] \rfloor$, for each $v_i \in \mathcal{N}_j^-$. It also sets the quantization value $c_{lj}[0] = 1$ and calculates $L_{lj} = \lceil l_{lj}/c_{lj}[0] \rceil$ and $U_{lj} = \lfloor u_{lj}/c_{lj}[0] \rfloor$, for each $v_l \in \mathcal{N}_j^+$. Then, it sets $\tilde{l}_{ji} = L_{ji}c_{ji}[0]$ and $\tilde{u}_{ji} = U_{ji}c_{ji}[0]$, for each $v_i \in \mathcal{N}_j^-$, and $\tilde{l}_{lj} = L_{lj}c_{lj}[0]$ and $\tilde{u}_{lj} = U_{lj}c_{lj}[0]$, for each $v_l \in \mathcal{N}_j^+$. If $\tilde{l}_{ji} > \tilde{u}_{ji}$ for some $v_i \in \mathcal{N}_j^-$ (or $\tilde{l}_{lj} > \tilde{u}_{lj}$ for some $v_l \in \mathcal{N}_j^+$); then it refines $c_{ji}[0] := c_{ji}[0]/2$ (or $c_{lj}[0] := c_{lj}[0]/2$) and recalculates L_{ji}, U_{ji}, $\tilde{l}_{ji}$, $\tilde{u}_{ji}$ (or L_{lj}, U_{lj}, $\tilde{l}_{lj}$, $\tilde{u}_{lj}$), and repeats if necessary.

The rest of the actions taken at initialization resemble those in Algorithm 1. In addition, each node v_j initializes value $vot_j[0] = 0$ if $b_j[0] \geq 0$, otherwise (if $b_j[0] < 0$) to $vot_j[0] = 1$; this value will be used in the max-consensus protocol and is not present in Algorithm 1.

Iteration. At each iteration $k \geq 0$, node v_j is aware of the flows on its incoming edges $\{f_{ji}[k] \mid v_i \in \mathcal{N}_j^-\}$ and outgoing edges $\{f_{lj}[k] \mid v_l \in \mathcal{N}_j^+\}$, and updates them using the following four steps.

[Step 1.] Step 1 is identical to Step 1 of Algorithm 1.

[Step 2.] Step 2 is identical to Step 2 of Algorithm 1.

[Step 3.] If value $\tilde{f}_{ji}[k+1]$ is in the interval $[\tilde{l}_{ji}, \tilde{u}_{ji}]$, then $f_{ji}[k+1] = \tilde{f}_{ji}[k+1]$; otherwise, if it is below $\tilde{l}_{ji}$ (respectively, above $\tilde{u}_{ji}$), it is set to the lower bound $\tilde{l}_{ji}$ (respectively, to the upper bound $\tilde{u}_{ji}$), i.e., the process is identical to (6). Then, it calculates $b_j[k+1]$ (based on the newly obtained flows) and sets

$$vot_j[k+1] = \begin{cases} 2, & \text{if } b_j[k] < b_j[k+1] < 0\,, \\ \max\{vot_j[k], 1\}, & \text{if } b_j[k+1] = b_j[k] < 0\,, \\ 0, & \text{if } b_j[k+1] \geq 0. \end{cases} \tag{7}$$

When the number of iterations is an integer multiple of $4m^2$ (i.e., $k \mod (4m^2) = 0$), every node v_j executes a max-consensus protocol for $n-1$ iterations, as described below.

[Step 4.] It sets $dec_j[0] = vot_j[k+1]$ and then for $k' = 0, 1 \ldots n-1$ it executes a max-consensus protocol by broadcasting $dec_j[k']$ to every out-neighbor, receiving $dec_i[k']$ from every in-neighbor and then setting $dec_j[k'+1] = \max_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} \{dec_i[k']\}$. One should point out that after $n-1$ iterations of the max-consensus protocol, the value $dec_j[n]$ at each node will be identical and equal to the maximum of the initial values, i.e.,

$$dec_j[n] = \max_{v_i \in \mathcal{V}} \{dec_i[0]\}, \text{ for all } v_j \in \mathcal{V}\,.$$

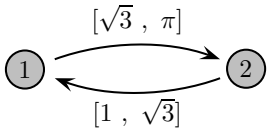

Fig. 1. Example of digraph with 2 nodes where flows and flow limits are not commensurable, and the lower and upper limits satisfy the circulation conditions in Theorem 1 with equality.

Based on the value of $dec_j[n]$, there are three possible cases to consider. [Since this value is the same for all nodes, all nodes execute one of the three cases below.]

Case 1. If $dec_j[n] = 0$, node v_j stops executing the proposed algorithm because flow balancing has been reached.

Case 2. If $dec_j[n] = 1$, then node v_j refines the flow limits of its incoming and outgoing links, i.e., it sets $c_{ji}[k+1] = c_{ji}[k]/2$ for every $v_i \in \mathcal{N}_j^-$ and $c_{lj}[k+1] = c_{lj}[k]/2$ for every $v_l \in \mathcal{N}_j^+$, and re-calculates

$$L_{ji} = \lceil l_{ji}/c_{ji} \rceil \text{ and sets } \tilde{l}_{ji} = L_{ji} c_{ji} \ , \quad (8)$$

$$U_{ji} = \lfloor u_{ji}/c_{ji} \rfloor \text{ and sets } \tilde{u}_{ji} = U_{ji} c_{ji} \ , \quad (9)$$

$$L_{lj} = \lceil l_{lj}/c_{lj} \rceil \text{ and sets } \tilde{l}_{lj} = L_{lj} c_{lj} \ , \quad (10)$$

$$U_{lj} = \lfloor u_{lj}/c_{lj} \rfloor \text{ and sets } \tilde{u}_{lj} = U_{lj} c_{lj} \ , \quad (11)$$

where $c_{ji} = c_{ji}[k+1]$ and $c_{lj} = c_{lj}[k+1]$. It also sets $vot_j[k] = 0$ if $b_j[k] \geq 0$, otherwise (if $b_j[k] < 0$) to $vot_j[k] = 1$, and goes to Step 1.

Case 3. If $dec_j[n] = 2$, node v_j performs no refinement, i.e., it sets $c_{ji}[k+1] = c_{ji}[k]$ for every $v_i \in \mathcal{N}_j^-$ and $c_{lj}[k+1] = c_{lj}[k]$ for every $v_l \in \mathcal{N}_j^+$, and does not execute (8)–(11). As in the previous case, node v_j resets $vot_j[k]$ to $vot_j[k] = 0$ if $b_j[k] \geq 0$, otherwise (if $b_j[k] < 0$) to $vot_j[k] = 1$, and goes to Step 1.

The pseudocode for the algorithm described above is provided as Algorithm 2.

Example 1: Consider the digraph in Fig. 1 in which flow limits are not commensurable and the circulation conditions are satisfied with equality. If we execute Algorithm 2, we observe that the algorithm is not able to reach a set of admissible and balanced flows after a finite number of iterations. However, if the upper limit on the flow from node 2 to node 1 is set equal to $\sqrt{3.05}$ (instead of $\sqrt{3}$), then we have flow limits that are not commensurable but the circulation conditions are satisfied with strict inequality. In such case, Algorithm 2 is able to reach a set of admissible and balanced flows after a finite number of iterations (after 179 iterations the absolute balance becomes equal to zero). Note that, in this example, $m = 2$ and therefore flow limit refinements can take place every $4m^2 = 16$ steps.

IV. Termination of Algorithm 2

A. Bounding the Termination Time of Algorithm 1

Setup 1. Consider the Flow Assignment Problem described in Section II, where we additionally assume that all flow limits are commensurable, i.e., for each edge $(v_j, v_i) \in \mathcal{E}_p$, we have

Algorithm 2: Enhanced Finite Time Distributed Flow Balancing

Each node $v_j \in \mathcal{V}$ has the following input/output.

Input: $l_{ji}, u_{ji}, \forall v_i \in \mathcal{N}_j^-$ and $l_{lj}, u_{lj}, \forall v_l \in \mathcal{N}_j^+$
Output: $f_{ji}, \forall v_i \in \mathcal{N}_j^-$ and $f_{lj}, \forall v_l \in \mathcal{N}_j^+$

Each node $v_j \in \mathcal{V}$ separately does the following.

Initialization (incoming): For every $v_i \in \mathcal{N}_j^-$
1. Set $c_{ji} = 1$
2. Calculate $L_{ji} = \lceil l_{ji}/c_{ji} \rceil$, $U_{ji} = \lfloor u_{ji}/c_{ji} \rfloor$, and set $\tilde{l}_{ji} = L_{ji} c_{ji}$, $\tilde{u}_{ji} = U_{ji} c_{ji}$
3. If $\tilde{l}_{ji} > \tilde{u}_{ji}$ set $c_{ji} := c_{ji}/2$ and go to Step 2

Initialization (outgoing): For every $v_l \in \mathcal{N}_j^+$
4. Set $c_{lj} = 1$
5. Calculate $L_{lj} = \lceil l_{lj}/c_{lj} \rceil$, $U_{lj} = \lfloor u_{lj}/c_{lj} \rfloor$, and set $\tilde{l}_{lj} = L_{lj} c_{lj}$, $\tilde{u}_{lj} = U_{lj} c_{lj}$
6. If $\tilde{l}_{lj} > \tilde{u}_{lj}$ set $c_{lj} := c_{lj}/2$ and go to Step 5

Execute Initialization Steps of Algorithm 1
7. Set $c_{ji}[0] = c_{ji}$ and $c_{lj}[0] = c_{lj}$
8. Set $vot_j[0] = 0$ if $b_j[0] \geq 0$, else set $vot_j[0] = 1$

Iteration: foreach $k = 0, 1, ...,$ **do**
- **Execute Iteration Steps of Algorithm 1**
- **Determine**: $b_j[k+1]$
- **Set**: $vot_j[k+1] =$
$$\begin{cases} 2, & \text{if } b_j[k] < b_j[k+1] < 0 \\ \max\{vot_j[k], 1\}, & \text{if } b_j[k+1] = b_j[k] < 0 \\ 0, & \text{if } b_j[k+1] \geq 0 \end{cases}$$
- **If** $k \mod (4m^2) \neq 0$ **Continue at Next Iteration**
- **Set**: $dec_j[0] = vot_j[k+1]$
- **foreach** $k' = 0, 1, ..., n-1$ **do**
  - Broadcast $dec_j[k']$ to every $v_l \in \mathcal{N}_j^+$
  - Receive $dec_i[k']$ from each $v_i \in \mathcal{N}_j^-$
  - Set $dec_j[k'+1] = \max_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} dec_i[k']$
- **If** $dec_j[n] = 0$ **End Operation**
- **Set** $vot_j[k] = 0$ if $b_j[k+1] \geq 0$, else set $vot_j[k] = 1$
- **If** $dec_j[n] = 1$
- **Set:** (i) $c_{ji}[k+1] = c_{ji}[k]/2$ for every in-neighbor $v_i \in \mathcal{N}_j^-$ and $c_{lj}[k+1] = c_{lj}[k]/2$ for every out-neighbor $v_l \in \mathcal{N}_j^+$
- (ii) for all $v_i \in \mathcal{N}_j^-$ and for all $v_l \in \mathcal{N}_j^+$ set: $L_{ji} = \lceil l_{ji}/c_{ji} \rceil$, $U_{ji} = \lfloor u_{ji}/c_{ji} \rfloor$, and $\tilde{l}_{ji} = L_{ji} c_{ji}$, $\tilde{u}_{ji} = U_{ji} c_{ji}$; $L_{lj} = \lceil l_{lj}/c_{lj} \rceil$, $U_{lj} = \lfloor u_{lj}/c_{lj} \rfloor$, and $\tilde{l}_{lj} = L_{lj} c_{lj}$, $\tilde{u}_{lj} = U_{lj} c_{lj}$, where $c_{ji} = c_{ji}[k+1]$ and $c_{lj} = c_{lj}[k+1]$

$l_{ji} = l'_{ji}c$ and $u_{ji} = u'_{ji}c$, where c is a positive real number, l'_{ji} is a nonnegative integer, and u'_{ji} is a positive integer.

The proof of the following proposition can be found in [20]. The proof of Theorem 2 can use a contradiction argument similar to the one in [22] (not developed here due to space limitations).

Proposition 1: Consider the problem formulation described in Section II. The absolute balance of the network at iteration k of Algorithm 1 (refer to Definition 2) satisfies

$$0 \leq \varepsilon[k+1] \leq \varepsilon[k] \ , \ \ \forall k \geq 0 \ .$$

Theorem 2: Consider *Setup* 1 described at the beginning of this section where the physical topology is given by digraph $\mathcal{G}_p = (\mathcal{V}, \mathcal{E}_p)$ with lower and upper flow limits on the edges such that the circulation conditions in Theorem 1 hold. Algorithm 1 reaches a set of feasible and balanced flows after a finite number of iterations.

In order to bound the execution time of Algorithm 1, we first bound the number of steps K it takes, in the worst case, for $\varepsilon[k+K]$ to become strictly smaller than $\varepsilon[k]$. We argue below in Theorem 3 that we can take $K = 4m^2$, where $m = |\mathcal{E}_p|$ is the number of edges of the given digraph; this implies that the total time to termination of Algorithm 1 satisfies

$$\text{Total Time} \leq 4m^2 \frac{\varepsilon[0]}{2c} \leq 2m^2 \Big(\sum_{j=1}^{n} \big| \sum_{v_i \in \mathcal{N}_j^-} l'_{ji} - \sum_{v_l \in \mathcal{N}_j^+} l'_{lj} \big| \Big) \ ,$$

where $\sum_{v_i \in \mathcal{N}_j^-} l'_{ji} - \sum_{v_l \in \mathcal{N}_j^+} l'_{lj}$ is the initial balance of node v_j divided by c. The reason is that every $K = 4m^2$ steps we get a decrease of at least $2c$ because a positive balance of $+c$ gets absorbed into a negative balance of $-c$, resulting in an improvement of $2c$ in absolute balance terms.

Theorem 3: Consider *Setup* 1 described at the beginning of this section where the physical topology is given by digraph $\mathcal{G}_p = (\mathcal{V}, \mathcal{E}_p)$ with lower and upper flow limits on the edges such that the circulation conditions in Theorem 1 hold. During the execution of Algorithm 1, we have

$$\varepsilon[k+K] < \varepsilon[k], \quad k = 0, 1, 2, ...$$

when $\varepsilon[k] > 0$ and $K \geq 4m^2$, where $m = |\mathcal{E}_p|$ is the number of edges in the given digraph $\mathcal{G}_p$).

Proof 1: Since $\varepsilon[k] > 0$, we have at least one node with positive balance, say node v_1, and at least one node with negative balance, say node v_n (all nodes, including node v_1 and node v_n will have balances that are commensurable, i.e., integer multiples of c). At each iteration of Algorithm 1, node v_n (in fact, any node with negative balance) will retain its negative balance unless at least one of its in-neighbors v_{n_i}, $(v_n, v_{n_i}) \in \mathcal{E}_p$ (and/or at least one of its out-neighbors v_{n_l}, $(v_{n_l}, v_n) \in \mathcal{E}_p$) has *positive* balance and increases (decreases) the flow on edge (v_n, v_{n_i}) $\big($edge $(v_{n_l}, v_n)\big)$. The reason is that node v_n has negative balance and does not attempt to change any of its incoming or outgoing flows. Also note that when one or more of node v_n's in-neighbors or out-neighbors have positive balance and successfully change one or more of its incoming or outgoing flows, it follows from the analysis in the proof of Theorem 2 in [20] that the absolute balance will decrease (by at least $2c$).

In order to determine a bound on the number of steps K required for the absolute balance to decrease, we can assume without loss of generality that negative nodes remain negative (because at the moment any negative node becomes balanced or positive, we also have a decrease by at least $2c$ in the absolute balance). We first consider the case when a single node, say v_1, has balance b (some positive integer multiple of c), and a single node, say v_n, has balance $-b$, and the remaining nodes v_2, v_3, ..., v_{n-1} are all balanced; then, we discuss the more general case.

At the first iteration, node v_1 transfers its balance b to one neighboring node (either an out-neighbor or an in-neighbor) by increasing/decreasing the flow on the corresponding outgoing/incoming edge. This neighbor of node v_1 does the same at the next iteration, and this process is repeated. If, at any point, node v_n is reached, the overall absolute balance will decrease by at least $2c$ (i.e., $\varepsilon[k+1] \leq \varepsilon[k] - 2c$). (For an illustration of the process, refer to Fig. 2, where v_1 is the node on the far left and v_n is the node on the far right.)

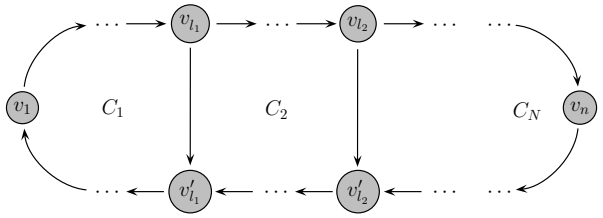

Fig. 2. Example of the transfer of positive balance from node v_1 on the left to node v_n on the right.

Let us now analyze the number of iterations it takes for node v_n to be reached. We make the following observations:
(i) At each iteration, the balance b gets transferred from a node to one of that node's in-neighbors or out-neighbors. We will say that this balance transfer process occurs by traversing an edge in the forward or in the backward direction (depending on whether the balance b gets transferred to an out-neighbor or to an in-neighbor). Note that there is a possibility that the balance transfer (i) is partial (because the node with positive balance attempts to change a flow in a way that exceeds its limits), or (ii) does not occur at all (i.e., the balance b remains at the same node, because the outgoing (incoming) edge chosen by the positively balanced node is already at its maximum (minimum) possible flow). In the former case, we are guaranteed that a balance transfer of at least c takes place, whereas in the latter case, this simply means that this particular edge has been used in the chosen direction without any effect (but we had to spend one step in attempting to change it).
(ii) As the iterations proceed, mark the first time an edge is traversed for the second time in the *same* direction, and call the cyclic sequence of edges visited up to this point C_1 (refer to Fig. 2). Note that C_1 is a cyclic sequence of edges (not nodes, i.e., a certain node may be visited more than once while traversing cycle C_1). Also note that C_1 has at most $2m$ edges because that is the total number of edges of the digraph, each associated with a forward and a backward direction.
(iii) While traversing C_1 for the second time, we will be forced at some point to traverse a new edge that has not been

traversed before (otherwise, since v_n has not been reached, the digraph is not strongly connected which is a contradiction). The reason is the fact that the ordering, chosen by each node at initialization on its outgoing/incoming edges, will force a different choice each time that node is visited. Let C_2 denote the set of edges traversed until the time we stop traversing new edges (i.e., we are forced to traverse an edge in a direction that we have already traversed, either in C_1 or in C_2).
(iv) We can continue in this fashion (defining C_i as shown in Fig. 2) until we reach node v_n (we are guaranteed to reach node v_n as long as the graph is strongly connected and the circulation conditions are satisfied). Note that the number of cycles N satisfies $N \leq 2m$, because there are $m = |\mathcal{E}_p|$ edges in the given digraph $\mathcal{G}$ (each considered twice, once in the forward direction and once in the backward direction) and each cycle has at least one edge. It also easy to see that node v_n will be reached without needing to traverse any cycle by more than N times. In particular, notice that, if cycle C is traversed k times, then any cycle C' that is an off-spring of C will be traversed $k-1$ times. (In an arrangement like the one in Fig. 2, cycle C_i will be traversed $N-i+1$ times.) In other words, we can upper bound the number of iterations to reach v_n by $N\sum_{i=1}^{N}|C_i| \leq (2m)^2$, where $|C_i|$ is the number of edges involved in cycle C_i and we have used the fact that $N \leq 2m$ and $\sum_{i=1}^{N}|C_i| \leq 2m$.

Note that the above analysis provides a upper bound K on the number of steps needed to guarantee a decrease (of at least $2c$) in $\varepsilon[k+K]$ compared to $\varepsilon[k]$, when we have a single node with positive balance and a single node with negative balance. If there are more than one nodes with positive balance and/or more than one nodes with negative balance, the algorithm simultaneously implements all actions taken by nodes with positive balance at any given step; thus, a decrease of $2c$ in the absolute balance will occur as soon as one node with positive balance manages to transfer (any part of) positive balance to a node with negative balance. This event cannot be delayed when multiple nodes with positive balance are present: the only way for the balance transfer process from node v_1 to node v_n not to occur in the same way as in Fig. 2 due to the presence of another positively balanced node is for the balance of that other node to reach and utilize a particular edge in the path at an earlier time step. In any case, a positive balance of at least c will reach node v_n within K steps. □

B. Establishing Termination in Finite Time for Algorithm 2

Setup 2. Consider the Flow Assignment Problem described in Section II, where we do not require that flow limits are commensurable.

Lemma 1: Consider *Setup* 2 described above where the physical topology is given by digraph $\mathcal{G}_p = (\mathcal{V}, \mathcal{E}_p)$ with lower and upper flow limits l_{ji} and u_{ji} on every edge $(v_j, v_i) \in \mathcal{E}_p$ such that the circulation conditions in Theorem 1 hold with strict inequality. Suppose that $\tilde{u}_{ji}$ and $\tilde{l}_{ji}$ are calculated, at time step k, according to $c_{ji}[k] = 2^{-\alpha_{ji}[k]}$ for some $\alpha_{ji}[k] \in \mathbb{N}_0$, using (8)–(11). Then, there exists an integer $\alpha_{\min}$ such that if $\alpha_{ji}[k] \geq \alpha_{\min}$ for all $(v_j, v_i) \in \mathcal{E}_p$, the following property holds: for each $\mathcal{S}$, $\mathcal{S} \subset \mathcal{V}$, we have

$$\sum_{(v_j,v_i)\in\mathcal{E}_{\mathcal{S}}^-} \tilde{l}_{ji} < \sum_{(v_l,v_j)\in\mathcal{E}_{\mathcal{S}}^+} \tilde{u}_{lj} \ , \tag{12}$$

where $\mathcal{E}_{\mathcal{S}}^-$ and $\mathcal{E}_{\mathcal{S}}^+$ are defined in (3) and (4), respectively.

Proof 2: Since the circulation conditions hold with strict inequality, this means that for each $\mathcal{S}$, $\mathcal{S} \subset \mathcal{V}$, (2) holds with strict inequality. Since the number of subsets $\mathcal{S}$ is finite, there exists $\varepsilon_{\min} > 0$ such that for each $\mathcal{S}$, $\mathcal{S} \subset \mathcal{V}$, we have

$$\varepsilon_{\min} < \sum_{(v_l,v_j)\in\mathcal{E}_{\mathcal{S}}^+} u_{lj} - \sum_{(v_j,v_i)\in\mathcal{E}_{\mathcal{S}}^-} l_{ji} \ . \tag{13}$$

During each iteration step k, the stricter upper and lower limits for each edge $(v_j, v_i) \in \mathcal{E}_p$ are calculated as $\tilde{u}_{ji}$ and $\tilde{l}_{ji}$ according to $c_{ji}[k] = 2^{-\alpha_{ji}[k]}$ for some $\alpha_{ji}[k] \in \mathbb{N}_0$, using (8)–(11). Notice that, for any $0 < \varepsilon < 1$, if $\alpha_{ji}[k] > \alpha$ where $\alpha = \lceil -\log_2(\varepsilon) \rceil$ then both of the following inequalities hold

$$0 \leq u_{ji} - \tilde{u}_{ji} \leq \varepsilon \ , \quad \text{and} \quad 0 \leq \ \tilde{l}_{ji} - l_{ji} \leq \varepsilon \ . \tag{14}$$

This means that for each $\mathcal{S}$, $\mathcal{S} \subset \mathcal{V}$, we have

$$0 \leq \sum_{(v_l,v_j)\in\mathcal{E}_{\mathcal{S}}^+} u_{lj} - \sum_{(v_l,v_j)\in\mathcal{E}_{\mathcal{S}}^+} \tilde{u}_{lj} \ \leq \ m\varepsilon \ , \tag{15}$$

$$0 \leq \sum_{(v_j,v_i)\in\mathcal{E}_{\mathcal{S}}^-} \tilde{l}_{ji} - \sum_{(v_j,v_i)\in\mathcal{E}_{\mathcal{S}}^-} l_{ji} \ \leq \ m\varepsilon \ , \tag{16}$$

where $m = |\mathcal{E}_p|$ is the number of edges.

Combining (15) and (16), we have

$$\Big(\sum_{(v_l,v_j)\in\mathcal{E}_{\mathcal{S}}^+} \tilde{u}_{li} - \sum_{(v_j,v_i)\in\mathcal{E}_{\mathcal{S}}^-} \tilde{l}_{ji} \Big) \ \geq$$
$$\Big(\sum_{(v_l,v_j)\in\mathcal{E}_{\mathcal{S}}^+} u_{li} - \sum_{(v_j,v_i)\in\mathcal{E}_{\mathcal{S}}^-} l_{ji} \Big) - 2m\varepsilon \tag{17}$$

From (14), (15) and (16), if we choose $\alpha \in \mathbb{N}_0$ such that $\varepsilon < \min\left(1, \frac{\varepsilon_{\min}}{2m}\right)$ (where $\varepsilon_{\min}$ is shown in (13)), we will ensure that for all $\mathcal{S}$, $\mathcal{S} \subset \mathcal{V}$, we have $\Big(\sum_{(v_l,v_j)\in\mathcal{E}_{\mathcal{S}}^+} \tilde{u}_{lj} - \sum_{(v_j,v_i)\in\mathcal{E}_{\mathcal{S}}^-} \tilde{l}_{ji} \Big) \geq 0$. In particular, the circulation conditions on the stricter limits will hold for $\alpha_{\min} = \max\left(0, \left\lceil -\log_2\left(\frac{\varepsilon_{\min}}{2m}\right)\right\rceil\right)$. □

The following theorem builds on this result and states that Algorithm 2 reaches a set of feasible and balanced flows after a finite number of iterations as long as the circulation conditions in Theorem 1 hold with strict inequality for the lower and upper flow limits on the edges.

Theorem 4: Consider *Setup* 2 described above where the physical topology is given by digraph $\mathcal{G}_p = (\mathcal{V}, \mathcal{E}_p)$ with lower and upper flow limits on the edges such that the circulation conditions in Theorem 1 hold with strict inequality. Algorithm 2 reaches a set of feasible and balanced flows after a finite number of iterations.

Proof 3: Algorithm 2 operates in phases of $4m^2$ steps each. During each phase, the stricter lower and upper flow limits are kept fixed and the algorithm essentially emulates the operation of Algorithm 1. Consider a phase that starts at iteration k and ends at iteration $k + 4m^2$. Let $c[k] = \min_{(v_j,v_i)\in\mathcal{E}_p}\{c_{ji}[k]\}$ (where $c_{ji}[k] = c_{ji}[k+1] = ... = c_{ji}[k+4m^2]$ for all $(v_j, v_i) \in \mathcal{E}_p$). We can easily establish the following.

1. During each phase, all flows and stricter flow limits are commensurable. More specifically, for the phase that starts at iteration k, all flows and stricter flow limits are multiples of $c[k]$ (this can be established easily by induction using the fact that any refinement at the end of the phase that starts at iteration k simply divides $c_{ji}[k]$ — and $c[k]$ — by a factor of 2; thus, at any given iteration k, flows and stricter flow limits are integer multiples of $c[k]$).
2. Assuming that the circulation conditions in Theorem 1 hold for the stricter flow limits on the edges at the beginning of the phase, we know from the proof of Theorem 3 that $\varepsilon[k+4m^2] \leq \varepsilon[k]-2c[k]$ as long as $\varepsilon[k] > 0$ (note that $\varepsilon[k]$ will necessarily be a multiple of $2c[k]$ by a positive integer). Moreover, in such case, at least one node v_j that had negative balance at the beginning of the iteration will increase its balance so that one of the following two conditions will hold:
Condition 1: $b_j[k] < b_j[k+4m^2] < 0$, or
Condition 2: $b_j[k] < 0 \leq b_j[k+4m^2]$.
If Condition 2 above holds for all nodes that were negative at the beginning of the phase, the result of the max consensus is 0 and signifies that flow balancing has been achieved (thus, Algorithm 2 can terminate). On the other hand, if Condition 1 holds for at least one node that had negative balance at the beginning of the phase, then the result of the max consensus will be 2; this indicates that there has been improvement in the absolute balance of the network and that the algorithm can continue to the next phase without refining the stricter lower/upper flow limits.
3. If the circulation conditions in Theorem 1 do not hold for the stricter lower and upper flow limits on the edges at the beginning of the phase, it is possible that we get Conditions 1 and/or 2 above (i.e., it is possible that there is an improvement of at least $2c[k]$ at the end of the phase). However, we will eventually run into a subsequent phase that starts at, say, iteration k', where all nodes that were negative at the beginning of the phase satisfy the following condition at the end of phase:
Condition 3: $b_j[k'] = b_j[k'+4m^2] < 0$ for all $v_j \in \mathcal{V}$.
Note that the above condition will hold after a finite number of subsequent phases because each phase leads to an improvement of an integer multiple of $2c[k]$; eventually, since the circulation conditions do not hold, there will be no improvement and all nodes with negative balance at the beginning of a phase will retain their negative balance (this follows because nodes with nonnegative balance retain nonnegative balance as argued after Proposition 1 in [20]). When Condition 3 holds, it will lead to a refinement of the stricter lower/upper flow limits because the result of the max consensus will be 1.
4. After a finite number of refinements, we are guaranteed from Lemma 1, that the circulations conditions will hold for the stricter lower/upper limits, at the beginning of a phase at iteration k'. From that point onward, each phase will lead to an improvement of at least $2c[k']$ (or a multiple of it by a positive integer), where $c[k']$ is the minimum value of $c_{ji}[k']$ at the beginning of the phase.

In summary, each phase leads to an improvement of the balance, unless the circulation conditions on the stricter lower/upper flow limits are not satisfied. In the latter case, the max-consensus outcome will lead to a refinement of the limits. Since improvements are non-diminishing between phases, refinements will occur after a finite number of steps. Moreover, there is a need for only a finite number of refinements. This implies that Algorithm 2 will complete after a finite number of iterations. □

REFERENCES

[1] L. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton University Press, 2010.
[2] R. T. Rockafellar, *Network Flows and Monotropic Optimization*. Belmont, Massachusetts: Athena Scientific, 1998.
[3] C. N. Hadjicostis, A. D. Domínguez-García, and T. Charalambous, "Distributed averaging and balancing in network systems, with applications to coordination and control," *Foundations and Trends® in Systems and Control*, vol. 5, no. 3–4, 2018.
[4] J. F. Mota, J. M. Xavier, P. M. Aguiar, and M. Püschel, "Distributed optimization with local domains: Applications in MPC and network flows," *IEEE Trans. on Automatic Control*, vol. 60, no. 7, pp. 2004–2009, 2014.
[5] G. Dantzig and D. R. Fulkerson, "On the max flow min cut theorem of networks," *Linear Inequalities and Related Systems*, vol. 38, pp. 225–231, 2003.
[6] T. Leighton and S. Rao, "An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms," Massachusetts Institute of Technology, Microsystems Research Center, Tech. Rep., 1989.
[7] H. H. Yang and D. Wong, "Efficient network flow based min-cut balanced partitioning," in *The Best of ICCAD*. Springer, 2003, pp. 521–534.
[8] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, and M. Reddy, "Applications of network optimization," *Handbooks in Operations Research and Management Science*, vol. 7, pp. 1–83, 1995.
[9] S. T. Cady, A. D. Domínguez-García, and C. N. Hadjicostis, "A distributed generation control architecture for islanded ac microgrids," *IEEE Trans. on Control Systems Technology*, vol. 23, no. 5, pp. 1717–1735, 2015.
[10] M. Zholbaryssov and A. D. Domínguez-García, "Convex relaxations of the network flow problem under cycle constraints," *IEEE Trans. on Control of Network Systems*, vol. 7, no. 1, pp. 64–73, 2019.
[11] G. Oliva, A. I. Rikos, C. N. Hadjicostis, and A. Gasparri, "Distributed flow network balancing with minimal effort," *IEEE Trans. on Automatic Control*, vol. 64, no. 9, pp. 3529–3543, 2019.
[12] P. DeLellis, M. di Bernardo, F. Garofalo, and M. Porfiri, "Evolution of complex networks via edge snapping," *IEEE Trans. on Circuits and Systems*, vol. 57, no. 8, pp. 2132–2143, 2010.
[13] W. Yu, P. DeLellis, G. Chen, M. di Bernardo, and J. Kurths, "Distributed adaptive control of synchronization in complex networks," *IEEE Trans. on Automatic Control*, vol. 57, no. 8, pp. 2153–2158, 2012.
[14] B. Gharesifard and J. Cortés, "Distributed strategies for generating weight-balanced and doubly stochastic digraphs," *European Journal of Control*, vol. 18, no. 6, pp. 539–557, 2012.
[15] A. D. Domínguez-García and C. N. Hadjicostis, "Distributed matrix scaling and application to average consensus in directed graphs," *IEEE Trans. on Automatic Control*, vol. 58, no. 3, pp. 667–681, 2013.
[16] A. Priolo, A. Gasparri, E. Montijano, and C. Sagues, "A decentralized algorithm for balancing a strongly connected weighted digraph," in *Proc. of American Control Conf. (ACC)*, 2013, pp. 6547–6552.
[17] A. Rikos, T. Charalambous, and C. N. Hadjicostis, "Distributed weight balancing over digraphs," *IEEE Trans. on Control of Network Systems*, vol. 1, no. 2, pp. 190–201, May 2014.
[18] C. N. Hadjicostis and A. D. Domínguez-García, "Distributed balancing in digraphs under interval constraints," in *Proc. of IEEE Conf. on Decision and Control (CDC)*, 2016, pp. 1769–1774.
[19] ——, "Distributed balancing of commodity networks under flow interval constraints," *IEEE Trans. on Automatic Control*, vol. 64, no. 1, pp. 51–65, 2018.
[20] C. N. Hadjicostis, A. D. Domínguez-García, and A. I. Rikos, "Finite-time distributed flow balancing," in *Proc. of 58th IEEE Conf. on Decision and Control (CDC)*, 2019, pp. 903–908.
[21] N. Lynch, *Distributed Algorithms*. San Mateo: CA: Morgan Kaufmann Publishers, 1996.
[22] A. I. Rikos and C. N. Hadjicostis, "Distributed balancing with constrained integer weights," *IEEE Trans. on Automatic Control*, vol. 64, no. 6, pp. 2553–2558, 2018.