Distrib. Comput. (2005)
DOI 10.1007/s00446-005-0130-y

REGULAR PAPER

Maurice Herlihy · Srikanta Tirthapura

Self-stabilizing smoothing and balancing networks

Received: 17 July 2003 / Accepted: 27 October 2004 / Published online: 29 December 2005

Abstract A *smoothing network* is a distributed data structure that accepts tokens on input wires and routes them to output wires. It ensures that however imbalanced the traffic on input wires, the numbers of tokens emitted on output wires are approximately balanced.

Prior work on smoothing networks always assumed that such networks were properly initialized. In a real distributed system, however, network switches may be rebooted or replaced dynamically, and it may not be practical to determine the correct initial state for the new switch. Prior analyses do not work under these new assumptions.

This paper makes the following contributions. First, we show that some well-known 1-smoothing networks, known as counting networks, when started in an arbitrary initial state (perhaps chosen by an adversary), remain remarkably smooth, degrading from 1-smooth to $(\log n)$-smooth, where n is the number of input/output wires. For the networks that we consider, we show that the above $(\log n)$ bound for the smoothness is tight.

Our second contribution is to show how any balancing network can be made self-stabilizing with the addition of local stabilization actions and state, which restore the network back to a "legal state" even if it starts out in an illegal state.

Keywords Smoothing networks · Counting networks · Self-stabilization

A preliminary version of this work appeared in the *Proceedings of The 23rd International Conference on Distributed Computing Systems, Providence, Rhode Island, USA.*

M. Herlihy
Computer Science Department, Brown University, Providence, RI, USA
E-mail: mph@cs.brown.edu

S. Tirthapura (✉)
Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, USA
E-mail: snt@iastate.edu

1 Introduction

A *k-smoothing network* is a distributed data structure that accepts tokens on input wires and routes them to output wires. It ensures that no matter how imbalanced the traffic on input wires, the numbers of tokens emitted on output wires are approximately balanced, lying within k of one another, where k is a constant, independent of the number of tokens that have entered the network.

Smoothing networks are well-suited for load-balancing applications where tokens represent requests for service. Clients send tokens to arbitrary input wires, and these tokens are routed to servers in such a way that all servers receive approximately the same number of tokens.

Prior work on smoothing networks has always assumed that such networks were properly initialized. In a real distributed system, however, network switches may be rebooted or replaced dynamically, and it may not be practical to determine the "correct" initial state for the new switch. Prior analyses do not apply under these assumptions.

The first contribution of this paper is to show that some well-known 1-smoothing networks, called *counting networks*, when started in an arbitrary initial state (perhaps chosen by an adversary), produce outputs that are remarkably smooth. In particular, the *bitonic* and the *periodic* counting networks of width n, when started in any of its $\Omega(2^n)$ possible initial states, will produce outputs such that numbers of tokens emerging on different output wires lie within $(\log n)$ of one another. We show that this bound is tight for both the bitonic and the periodic networks.

In some cases, such as in distributed counting, the above $(\log n)$-smooth behavior under faults may not be sufficient for the application concerned. We do not know of prior work which provides a systematic way to recover from such faults in smoothing and counting networks. In this paper, we explore one possible approach to recovery from faulty states using *self-stabilization*.

The second contribution of this paper is to show that any balancing network can be made self-stabilizing, so that even if the network is started in an arbitrary state, it eventually

converges to a "legal" state (a precise definition of a legal state will follow). We analyze the time and the additional space required for stabilization. The information required for self-stabilization can be piggy-backed on existing messages. Since smoothing and counting networks are instances of balancing networks, our technique can be used to construct self-stabilizing smoothing and counting networks.

1.1 Balancing networks

A *balancer* is an asynchronous switch with two input wires and two output wires. The output wires are labeled 0 and 1. A balancer accepts a stream of tokens on its input wires. A balancer has two states: it is either oriented *up* or *down*. If the balancer is oriented *up*, then the next input token leaves on output wire 0, and the balancer becomes oriented *down*. If, however, the balancer is oriented *down*, then the next input token leaves on output wire 1, and the balancer becomes oriented *up*. *Reversing* the orientation of a balancer means changing its state from *up* to *down*, and vice-versa. If s is a balancer state, then $\bar{s}$ denotes the opposite state. We say a balancer is *oriented toward* wire i if the next token would leave on wire i.

A *balancing network* is an acyclic network of balancers where the output wires of some balancers are linked to the input wires of others. The *standard* initial state of a balancing network is one where all balancers are oriented *up*. An example balancing network is shown in Fig. 1, and different initializations of the same network in Fig. 2.

The network's *input wires* are those wires not linked to the output of any balancer, and similarly the network's *output wires* are those wires which are not linked to the input of any balancer. Unless specified otherwise, we consider balancing networks with the same number of input and output wires, called the network's *width*. Tokens enter the network on the input wires, typically several per wire, propagate asynchronously through the balancers, and leave on the output wires, typically several per wire. A state of a balancing network is *quiescent* if every token that has entered the network has also left it.

Because balancing networks are acyclic (as directed graphs), each balancer can be assigned a unique *layer*, which is the length of the longest path from an input wire to that balancer. This assignment is useful because it means that we can use induction on the number of layers to reason about balancing networks.

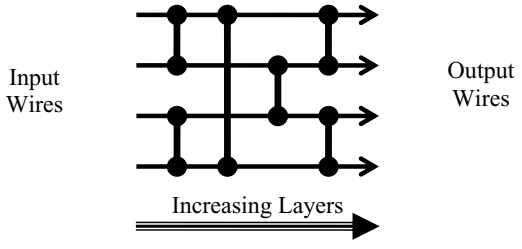

Fig. 1 A balancing network of width 4. Horizontal segments are the wires and the vertical segments are the balancers

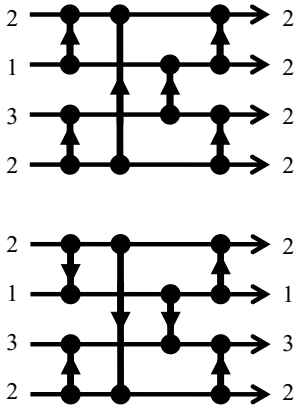

Fig. 2 Top and bottom are the same balancing network initialized in standard and non-standard states respectively. The numbers on the input/output wires indicate the numbers of tokens

A sequence $X = x_0, x_1 \ldots x_{n-1}$ is *k-smooth* if $|x_i - x_j| \leq k$, for any $0 \leq i, j < n$. A balancing network of width n is a *k-smoothing network* if, starting from its initial state, the following is true in any quiescent state of the network: the sequence $x_0, x_1, \ldots x_{n-1}$, where x_i is the number of tokens that have exited on output wire i, is a k-smooth sequence. A *counting network* [2] is a 1-smoothing network with the additional "step property": in a quiescent state, the excess tokens appear on the uppermost output wires.

1.2 Our contributions

We ask some new questions about smoothing networks. Prior work on smoothing networks has always assumed that networks are initialized in their standard initial states. We first ask the following question.

How do these networks behave if they can be initialized in any state, perhaps one chosen by an adversary?

The motivation for this question is as follows. Consider a distributed load-balancing network overlaid on a local area network. If a switch crashes and needs to be reset, or if one switch replaces another, then it is difficult to determine the "right" state for the new switch, and it is not practical to reinitialize the entire network. How badly will the network perform if we reset the switch to an arbitrary state?

We show the following results in response to the above question:

- The well-known *bitonic* and *periodic* counting networks are remarkably smooth even when not initialized to the standard initial state: a network of width n is $\log(n)$-smooth when started in any of its $2^{(\Theta(n \log^2 n))}$ possible initial states.
- The above bound is tight for the bitonic and periodic networks. We demonstrate initial states and input sequences such that the output sequences are not k-smooth for any $k < \log n$, where n is the width of the network.

Our results imply that if $\log n$-smoothness is enough for an application, then there is no need to reset these balancers

to the "correct" state upon a fault. Any network state will do. On the other hand, our analysis shows that for the *bitonic* and the *periodic* networks, the output smoothness might be no better than $\log n$. What if the above $\log n$-smoothness is not enough for an application? In such a case, we need a mechanism for the network to recover from faulty states. We explore a solution to this based on self-stabilization.

Self-Stabilization: We show how any balancing network can be made self-stabilizing [5] with the addition of local stabilization actions and state. Suppose we are given a desired behavior of a distributed system in the form of precisely defined "legal" and "illegal" states. The system is said to be self-stabilizing if, when started in an arbitrary state (perhaps illegal), it will eventually converge to a legal state and henceforth remain in a legal state.

If the balancing network reaches an "illegal" state during execution, then these self-stabilizing actions bring it back to a "legal" state (legal and illegal states of a balancing network are defined precisely in Sect. 5). On the other hand, if the network is in a legal state, then the self-stabilization actions are dormant. A key feature of self-stabilization is that no external action is needed to initiate the recovery from faults.

These stabilizing actions are local, that is, they involve interaction between neighboring balancers in the network. In addition, the stabilizing actions can take place in parallel with the normal execution of the network, and can also be "piggy-backed" on the regular tokens that pass through the network.

We analyze the time and the space overhead due to the stabilization. For networks of width n, if d denotes the depth of the network, then the time to stabilization is $O(d)$ parallel time steps and the additional space required for stabilization is $O(nd^2)$ bits. We present our self-stabilization and prove its correctness in Sect. 5.

1.3 Related work

Counting networks were introduced by Aspnes, Herlihy and Shavit [2] as a solution to the distributed counting problem. They described the bitonic and the periodic counting networks, which were isomorphic to correspondingly named sorting networks, the bitonic sorting network [3], and the periodic sorting network [7]. Both these had a depth of $O(\log^2 n)$ where n is the width of the network. There has been much work about balancing networks after that, and we refer the reader to the survey by Busch and Herlihy[4].

Riedel and Bruck [9] study the fault tolerance of counting networks. They propose adding a "correction network" at the end of a counting network. To tolerate k bit faults, the depth of their correction network is $k^2 \log n$, and the resulting sequence has a smoothness error of k. In contrast, we show that the output of some of the popular counting networks is always $\log n$ smooth, even if arbitrarily initialized and without any need for a correction layer.

Aiello, Venkatesan and Yung [1] build counting networks using "randomized balancers" which do not require any initialization. However, their construction also used some deterministic balancers which need to be initialized correctly. Their focus is mainly on building counting networks and not on smoothing networks.

Motivated by the question of recovery from faults, Herlihy and Tirthapura [8] recently study the behavior of smoothing networks whose balancers are initialized to a random state, rather than by an adversary. After a fault, it is easy for a balancer to set itself to a random initial state in a local manner. It was found that randomly initialized networks were much smoother than networks that were adversarially initialized. In particular, the Block balancing network of width n is $O(\sqrt{\log n})$ smooth when started from a random initial state, while in this paper we show that the smoothness may be no better than $O(\log n)$ if it was initialized adversarially.

Self-stabilizing distributed systems were first studied by Dijkstra [5]. A survey of results on self-stabilization can be found in [10], and a recent book on the topic has appeared [6].

Organization of the paper: The rest of this paper is organized as follows. Section 2 proves some general smoothing properties for balancing networks. Sections 3 and 4 analyze the smoothing properties of the Bitonic and the Periodic networks respectively. In Sect. 5 we present self-stabilization of balancing networks.

2 Smoothing properties of balancing networks

In this section, we prove some general smoothing properties for balancing networks. We denote sequences of natural numbers in upper case, and elements of a sequence in lower case. For example, a sequence $X = x_0, \ldots, x_{n-1}$ has length n. For brevity, we sometimes refer to a sequence of length n as an *n-sequence*.

Sequence X is *k-smooth* if $|x_i - x_j| \leq k$, for any $0 \leq i, j < n$. The elements of a k-smooth sequence take values in the range $a, a+1, \ldots, a+k$ for some a. A balancing network of width n is a k-smoothing network if, starting from its initial state, the following is true in any quiescent state of the network: the sequence $x_0, x_1, \ldots x_{n-1}$, where x_i is the number of tokens that have exited on output wire i, is a k-smooth sequence.

Sequence X possesses the *step* property if $0 \leq (x_i - x_j) \leq 1$ for any $0 \leq i < j < n$. A counting network [2] of width n has the following property. Starting from its initial state, the following is true in any quiescent state of the network: the sequence $x_0, x_1, \ldots x_{n-1}$, where x_i is the number of tokens that have exited on output wire i, has the step property. Note that the step property is stronger than 1-smoothness, and hence every counting network is a 1-smoothing network, though not vice versa.

If X and Y are n-sequences, $X \cdot Y$ denotes their concatenation: the $2n$-sequence $x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}$. If X is an n-sequence, $X + c$ denotes the n-sequence $x_0 + c, x_1 +$

$c, \ldots$. A *constant* sequence is one in which all elements are equal.

Let X be an n-sequence. For all n-sequences considered in this paper, n is a power of 2.

- X^R is the reverse n-sequence: $x_i^R = x_{n-i-1}$.
- X^E is the even $(n/2)$-subsequence: $x_i^E = x_{2i}$.
- If $n > 1$ then X^O is the odd $(n/2)$-subsequence: $x_i^O = x_{2i+1}$.

We will make use of the following facts:

Fact 1

$$(X \cdot Y)^R = Y^R \cdot X^R.$$

Fact 2 *If X and Y both have even length,*

$$(X \cdot Y)^E = X^E \cdot Y^E$$

Fact 3 *If X and Y both have even length,*

$$(X \cdot Y)^O = X^O \cdot Y^O$$

All sequences considered in this paper, except for sequences of length 1, have even length.

If X and Y are n-sequences, then $Z = U|V$, the *meld* of X and Y, is the n-sequence:

$$z_i = \begin{cases} x_i & \text{if } i \text{ is even,} \\ y_i & \text{if } i \text{ is odd.} \end{cases}$$

Let NETWORK$[n]$ be any balancing network of width n. A *state* $\mathcal{N}[n]$ for NETWORK$[n]$ is given by choosing a state (*up* or *down*) for each of the network's balancers. The *standard* state for any balancing network is the one where all balancers are in the *up* state. For brevity, we sometimes refer to the network $\mathcal{N}[n]$ to mean NETWORK$[n]$ initialized to state $\mathcal{N}[n]$. We sometimes omit explicit mention of n.

Let $X = x_0, \ldots, x_{n-1}$ be a sequence, and $\mathcal{N}$ an initialized network of width n. Suppose for each $i = 0 \ldots n-1$, we place x_i tokens on input wire i of $\mathcal{N}$, and run the tokens through the network until it becomes quiescent. The path of any individual token depends on the order in which the tokens' moves are interleaved. Nevertheless, once the network reaches a quiescent state, the output sequence is independent of the interleaving (proof in Aspnes et al. [2]). The network's initial state $\mathcal{N}$ thus defines a map carrying any n-sequence X to a well-defined n-sequence $\mathcal{N}(X)$.

Let X and Y be k-smooth n-sequences. A *matching* layer of balancers for X and Y is one where each element of X is joined by a balancer to an element of Y in a one-to-one correspondence. An example is shown in Fig. 3.

Lemma 1 *If X and Y are each k-smooth, and Z is the result of matching X and Y, then Z is $(k+1)$-smooth.*

Proof Let x and y be the smallest values in X and Y. Suppose a balancer joins x_i and y_j to produce z_i and z_j. If the balancer is oriented toward i, then

$$z_i = \left\lceil \frac{x_i + y_j}{2} \right\rceil \qquad \text{and} \qquad z_j = \left\lfloor \frac{x_i + y_j}{2} \right\rfloor.$$

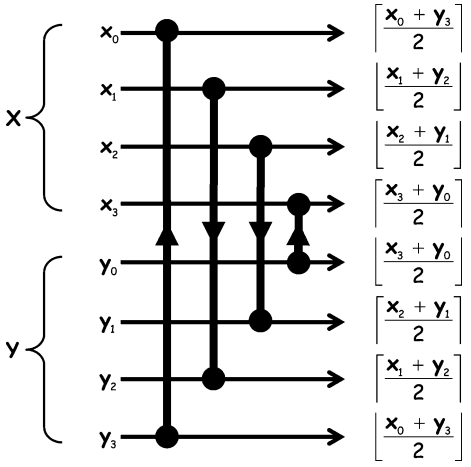

Fig. 3 A matching layer for X and Y.

The smallest value that any element in Z can assume is thus

$$\left\lfloor \frac{x+y}{2} \right\rfloor$$

and the largest value is

$$\left\lceil \frac{x+k+y+k}{2} \right\rceil = \left\lceil \frac{x+y}{2} \right\rceil + k \leq \left\lfloor \frac{x+y}{2} \right\rfloor + k + 1.$$

The largest and smallest elements of Z differ by at most $k+1$.

Lemma 2 *If X is k-smooth, then the result of feeding X through a layer of balancers is k-smooth.*

Proof We show that the result of passing X through one layer of balancers is still k-smooth, and the lemma follows by induction.

Let Z be the result of feeding X through a layer of balancers. The smallest element in Z is greater than or equal to the smallest element in X, and the largest element in Z is lesser than or equal to the largest element in X. Since X is k-smooth, Z is also k-smooth.

X is a *fixed point* sequence for a balancing network $\mathcal{N}$ if $\mathcal{N}(X) = X$.

Lemma 3 *Let $\mathcal{N}$ be any network state, and C the constant sequence in which each entry is c. Then, (1) C is a fixed point for $\mathcal{N}$, and (2) after C passes through, the network will return to quiescent state $\mathcal{N}$.*

Proof Suppose c is 1. One token starts out on each input wire. A simple induction on the number of balancers shows that one token emerges on each output wire. Each balancer is visited by exactly two tokens, leaving that balancer in its original state, and leaving the network in its original state $\mathcal{N}$.

For $c > 1$, the execution involving c tokens on each input wire can be viewed as a concatenation of c smaller executions, in each of which one token enters each input wire.

X, Y, Z denote sequences and $\mathcal{N}, \mathcal{M}, \mathcal{B}$ denote network states.
Let $X = x_0, x_1, \ldots, x_{n-1}$, $Y = y_0, y_1, \ldots, y_{n-1}$
$X^R = x_{n-1}, x_{n-2}, \ldots, x_0$
$X^E = x_0, x_2, x_4, \ldots$
$X^O = x_1, x_3, \ldots$
$X + c = x_1 + c, x_2 + c, \ldots, x_{n-1} + c$
$X|Y = x_0, y_1, x_2, y_3, \ldots$
$X \cdot Y = x_0, x_1, \ldots, x_{n-1}, y_0, y_1, \ldots, y_{n-1}$
$\mathcal{N}(X)$ is the output sequence resulting from passing X through $\mathcal{N}$
$\mathcal{N}^R$ denotes the reflection of state $\mathcal{N}$

Fig. 4 Summary of notation used in this paper

From the above argument, at the end of each such "smaller" execution, the network is back in state $\mathcal{N}$ and one token has exited each output wire. Thus, upon passing C through $\mathcal{N}$, the output is always C and the network is back to state $\mathcal{N}$.

Lemma 4 *If X is a fixed-point sequence for $\mathcal{N}$, then so is $X + c$ for any constant c.*

Proof Consider the execution in which c tokens enter each input wire and traverse the network. When the network becomes quiescent, Lemma 3 implies that c tokens have emerged on each output wire, and the network state returns to $\mathcal{N}$. Now run the the sequence X through the network. Since X is a fixed point for $\mathcal{N}$, the network will quiesce with output sequence $X + c$.

3 The bitonic network

The BITONIC[n] counting network [2] is isomorphic to the *Bitonic* sorting network of Batcher [3]. This network has a simple inductive structure. The BITONIC[2] network is a single balancer. As illustrated in Fig. 5, the BITONIC[$2n$] network is constructed by feeding the $2n$ input wires into two parallel BITONIC[n] networks, and feeding their outputs into a MERGER[$2n$] network.

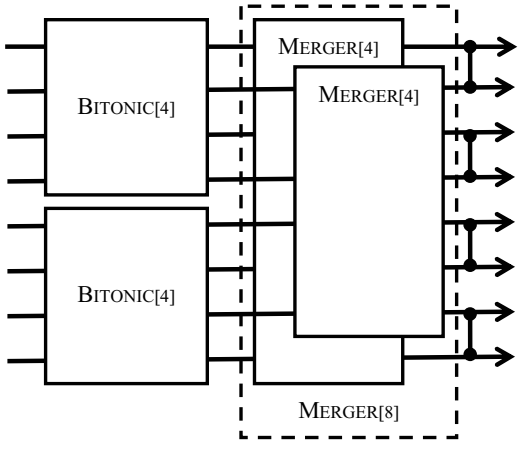

Fig. 5 Recursive structure of a BITONIC[8] counting network.

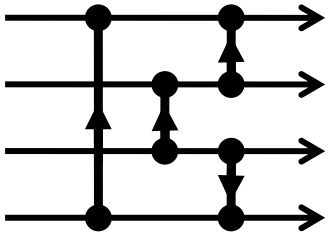

Fig. 6 The $\mathcal{M}^*$[4] initial state

The MERGER[$2n$] balancing network takes two input n-sequences X and Y and produces an output $2n$-sequence Z. The MERGER[n] network is also defined inductively. The MERGER[2] network is a single balancer. We construct the MERGER[$2n$] network from two MERGER[n] networks and a EVENODD[$2n$] network to be described.

The input to the first MERGER[n] network is the n-sequence $X^E \cdot Y^O$ (Fig. 4 contains a guide to the notation). Call that network's output sequence U. Symmetrically, the input to the second MERGER[n] network is the n-sequence $X^O \cdot Y^E$. Call that output sequence V. The final layer of the network, called EVENODD[n], simply joins each output wire of the first MERGER[n] network with the corresponding output wire of the second MERGER[n] network (as shown in Fig. 5).

Figure 6 shows a MERGER[4] network (ignore the arrowheads for now). Figure 7 shows how a MERGER[8] network is constructed from two parallel MERGER[4] networks (one in black and one in gray) feeding into a EVENODD[8] network (outlined by a dotted box).

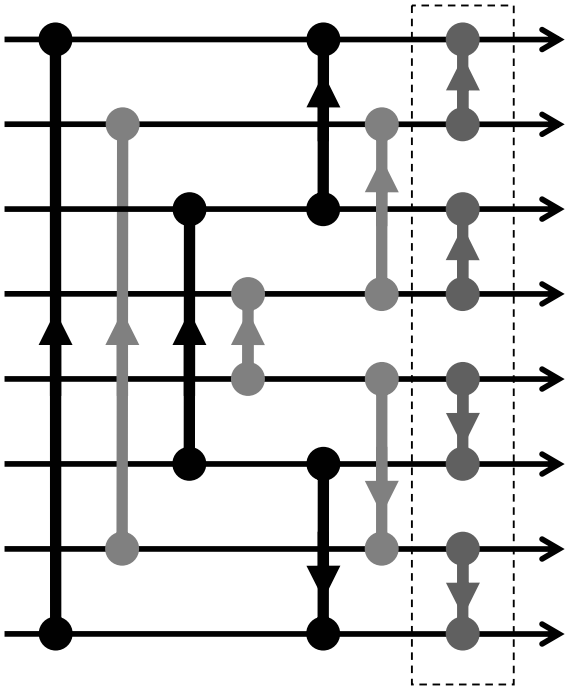

Fig. 7 The $\mathcal{M}^*$[8] initial state

3.1 Upper bound

In a BITONIC[$2n$] network in any state, the two component BITONIC[n] networks have output sequences X and Y, and the MERGER[$2n$] component has input $X \cdot Y$.

Lemma 5 *The first layer of the* MERGER[$2n$] *network is a matching between X and Y.*

Proof By induction. The base case, when $n = 2$, is trivial. As the induction hypothesis, assume that the first layer of MERGER[n] is a matching.

The first layer of MERGER[$2n$] is just the first layer of both MERGER[n] components. By the induction hypothesis, the first layer of these components are matchings for X^E and Y^O, and also for X^O and Y^E. Together they form a matching for X and Y.

The following theorem gives an upper bound on the smoothness of the BITONIC[n] network.

Theorem 1 *The output of a* BITONIC[n] *counting network initialized in any state is* $(\log n)$*-smooth.*

Proof By induction on n. When n is 2, the network is a single balancer, and the output is 1-smooth.

In the BITONIC[n] network, the two output $(n/2)$-sequences X and Y from the component BITONIC[$n/2$] networks are each $(\log n - 1)$-smooth by the induction hypothesis. These sequences are fed to a MERGER[n] network, whose first layer is a matching for X and Y by Lemma 5. Lemma 1 and Lemma 2 together imply that the output sequence Z is $\log n$-smooth.

3.2 Lower bound

We now show a lower bound on the smoothness of the BITONIC[n] network, showing that there exists an initial state and input sequence such that the output sequence is not k-smooth for any $k < \log n$.

A balancing network is *symmetric* if reflecting it around a horizontal axis yields the same network. More formally, the network topology is unchanged if we relabel wire i in every layer to be $n-i-1$, for $0 \le i \le n-1$. The BITONIC[n] network is symmetric.

Let $\mathcal{N}$ be an initial network state. Define $\mathcal{N}^R$ to be the network state constructed by "reflecting" $\mathcal{N}$ around a horizontal axis: if a balancer in $\mathcal{N}$ between wires i and j in level ℓ has state s, then the balancer in $\mathcal{N}^R$ between wires $n-i-1$ and $n-j-1$ in level ℓ has state $\bar{s}$.

Lemma 6 *Let $\mathcal{N}$ be an arbitrary initial state of a symmetric network. If $\mathcal{N}(X) = Y$, then $\mathcal{N}^R(X^R) = Y^R$.*

Proof By induction on the depth of the network. Let $\bar{Y} = \mathcal{N}^R(X^R)$. For the base case, consider a single layer. If wire i is not connected to a balancer, then because B is symmetric, then neither is wire $n-i-1$. In that case, $y_i = x_i = x^R_{n-i-1} = \bar{y}_{n-i-1}$. If a balancer b connects wire i with wire j, then because B is symmetric, a balancer b' also connects wire $n-i-1$ with wire $n-j-1$. If b is oriented toward i,

$$y_i = \left\lceil \frac{x_i + x_j}{2} \right\rceil = \left\lceil \frac{x^R_{n-i-1} + x^R_{n-j-1}}{2} \right\rceil = \bar{y}_{n-i-1},$$

and similarly if b is oriented toward j. As a result, $\bar{Y} = Y^R$.

For the induction step, assume the claim for networks of depth $d > 1$. The network $\mathcal{N}$ can be written as a network $\mathcal{N}_0$ of depth $d-1$ followed by a single-layer network $\mathcal{N}_1$. Let $Z = \mathcal{N}_0(X)$, and $Y = \mathcal{N}_1(Z)$. By the induction hypothesis, $Z^R = \mathcal{N}_0^R(X^R)$, and $Y^R = \mathcal{N}_1^R(Z^R) = \mathcal{N}^R(X^R)$.

Let $\mathcal{M}$ be any initial state of the MERGER[n] component.

Lemma 7 *If $\mathcal{M}(X) = Y$, then $\mathcal{M}^R(X) = Y^R$.*

Proof We split $\mathcal{M}[n]$ network into two stages. The first, $\mathcal{M}_0$, is just the first layer of $\mathcal{M}[n]$, and the second, $\mathcal{M}_1$ consists of the remaining layers. Let $\mathcal{M}_0(X) = Z$, $\mathcal{M}_0^R(X) = \bar{Z}$, and $\mathcal{M}_1(Z) = Y$.

In $\mathcal{M}_0$, balancer b_i joins i and $n-i-1$, for $0 \le i < n/2$.

$$z_i = \begin{cases} \left\lceil \frac{x_i + x_{n-i-1}}{2} \right\rceil & \text{if } b_i \text{ is oriented toward } i \\ \left\lfloor \frac{x_i + x_{n-i-1}}{2} \right\rfloor & \text{if } b_i \text{ is oriented toward } n-i-1 \end{cases}$$

$\mathcal{M}_0^R$ simply reverses the orientation of the balancers.

$$\bar{z}_{n-i-1} = \begin{cases} \left\lceil \frac{x_i + x_{n-i-1}}{2} \right\rceil & \text{if } b_i \text{ is oriented toward } i \\ \left\lfloor \frac{x_i + x_{n-i-1}}{2} \right\rfloor & \text{if } b_i \text{ is oriented toward } n-i-1 \end{cases}$$

Because $z_i = \bar{z}_{n-i-1}$, it follows that $\bar{Z} = Z^R$. Lemma 6 implies that $\mathcal{B}_1^R(Z^R) = (\mathcal{B}_1(Z))^R = Y^R$

Let $\mathcal{B}$ be any initial state of the BITONIC[n] network, and let $\mathcal{M}$ be the induced state of the component MERGER[n] network. Define $\mathcal{B}^*$ to be the initial state constructed from $\mathcal{B}$ by replacing $\mathcal{M}$ with $\mathcal{M}^R$.

Lemma 8 *If $\mathcal{B}(X) = Y$, then $\mathcal{B}^*(X) = Y^R$.*

Proof Split the BITONIC[n] network into two stages. The first state consists of the two parallel BITONIC[$n/2$] networks and the second stage is the MERGER[n] network. Let $\mathcal{B}_0$ be the first stage's state in $\mathcal{B}$, and $\mathcal{M}$ the second's. It follows that $\mathcal{B}_0$ is the first stage's state in $\mathcal{B}^*$, and $\mathcal{M}^R$ the second's. Let $Z = \mathcal{B}_0(X)$, so $Y = \mathcal{M}(Z)$. By Lemma 7, $\mathcal{M}^R(Z) = Y^R$, and so $\mathcal{B}^*(X) = \mathcal{M}^R(Z) = Y^R$.

Recall that $U|V$ is the meld of U and V. Let $\mathcal{E}$ denote the initial state of the EVENODD[n] network in which all balancers are initialized to *up*.

Lemma 9 *If $W = \mathcal{E}(U|V)$, then $W^R = \mathcal{E}^R(U^R|V^R)$.*

Proof Let $\bar{W} = \mathcal{E}^R(U^R|V^R)$.

$$w_i = \begin{cases} \lceil \frac{u_i+v_i}{2} \rceil & \text{if } i \text{ is even} \\ \lfloor \frac{u_i+v_i}{2} \rfloor & \text{if } i \text{ is odd.} \end{cases}$$

$$\bar{w}_i = \begin{cases} \lfloor \frac{u_{n-i-1}+v_{n-i-1}}{2} \rfloor & \text{if } i \text{ is even} \\ \lceil \frac{u_{n-i-1}+v_{n-i-1}}{2} \rceil & \text{if } i \text{ is odd.} \end{cases}$$

It follows that $\bar{w}_{n-i-1} = w_i$, and hence $\bar{W} = W^R$.

We now define a new state $\mathcal{M}^*[n]$ for the MERGER$[n]$ network. As illustrated in Fig. 6, in $\mathcal{M}[4]$, all balancers are oriented *up*, except the last lower balancer. We construct $\mathcal{M}^*[2n]$ as follows: initialize both component MERGER$[n]$ subnetworks to $\mathcal{M}^*[n]$. The EVENODD$[2n]$ network is initialized as follows: balancers 0 through $n-1$ are initialized to $\mathcal{E}[n]$ (all *up*), while balancers n to $2n-1$ are initialized to $\mathcal{E}^R[n]$ (all *down*). Fig. 7 shows a $\mathcal{M}^*[8]$ network, where one component $\mathcal{M}^*[4]$ is shown in black, the other in gray, and the final EVENODD$[8]$ is highlighted by a dotted box.

Lemma 10 *If X is an n-sequence and Y, Z are $n/2$-sequences such that $\mathcal{M}[n](X) = Y \cdot Z$, then $\mathcal{M}^*[n](X) = Y \cdot Z^R$.*

Proof We argue by induction on n. When n is 4, the claim can be verified by inspection. For the induction step, assume the result for n.

First, consider $\mathcal{M}[2n]$. Let the output of the first component $\mathcal{M}[n]$ network be $U \cdot V$, and the second $\bar{U} \cdot \bar{V}$. The output sequence on wires 0 to $n-1$ is $Y = \mathcal{E}(U|\bar{U})$, and on wires n to $2n-1$ is $Z = \mathcal{E}(V|\bar{V})$.

Next, consider $\mathcal{M}^*[2n]$. By the induction hypothesis, the output of the first component $\mathcal{M}^*[n]$ network is $U \cdot V^R$, and the second $\bar{U} \cdot \bar{V}^R$. The output sequence on wires 0 to $n-1$ is $Y = \mathcal{E}(U|\bar{U})$, and on wires n to $2n-1$ is $\bar{Z} = \mathcal{E}^R(V^R|\bar{V}^R)$. By Lemma 9, $\bar{Z} = Z^R$.

We are now ready to start constructing the fixed-point sequences. Consider the following sequences:

$$\begin{aligned} B_2 &= 1, 0 \\ B_{2n} &= (B_n + 1) \cdot B_n \\ C_{2n} &= (B_n + 1) \cdot B_n^R \end{aligned}$$

Notice that B_n is $(\log n)$-smooth.

Lemma 11 $B_{2n}^O = B_n$.

Proof By induction on n. For the base case, $B_2 = 1, 0$, $B_4 = 2, 1, 1, 0$, and $B_4^O = 1, 0 = B_2$. For the induction step,

$$\begin{aligned} B_{2n}^O &= (B_n + 1)^O \cdot B_n^O && \text{Fact 3} \\ &= (B_{n/2} + 1) \cdot B_{n/2} && \text{induction hypothesis} \\ &= B_n \end{aligned}$$

Lemma 12 $B_{2n}^E = B_n + 1$.

Proof By induction on n. For the base case, $B_2 = 1, 0$, $B_4 = 2, 1, 1, 0$, and $B_4^E = 2, 1 = B_2 + 1$. For the induction step,

$$\begin{aligned} B_{2n}^E &= (B_n + 1)^E \cdot B_n^E && \text{Fact 2} \\ &= (B_{n/2} + 2) \cdot (B_{n/2} + 1) && \text{induction hypothesis} \\ &= ((B_{n/2} + 1) \cdot B_{n/2}) + 1 \\ &= B_n + 1 \end{aligned}$$

Lemma 13 $(B_{2n}^R)^O = (B_n + 1)^R$.

Proof By induction on n. For the base case, $B_2^R = 0, 1$, $B_4^R = 0, 1, 1, 2$, and $(B_4^R)^0 = 1, 2 = (B_2 + 1)^R$. For the induction step,

$$\begin{aligned} (B_{2n}^R)^O &= ((B_n^R \cdot (B_n + 1)^R)^O && \text{Fact 1} \\ &= ((B_n^R)^O \cdot ((B_n + 1)^R)^O && \text{Fact 3} \\ &= (B_{n/2} + 1)^R \cdot (B_{n/2} + 2)^R && \text{Induction hypothesis} \\ &= ((B_{n/2} + 2) \cdot (B_{n/2} + 1))^R && \text{Fact 1} \\ &= ((B_{n/2} + 1) \cdot B_{n/2}) + 1)^R && \text{Fact 1} \\ &= (B_n + 1)^R \end{aligned}$$

Lemma 14 $(B_{2n}^R)^E = B_n^R$.

Proof By induction on n. For the base case, $B_2^R = 0, 1$, $B_4^R = 0, 1, 1, 2$, and $(B_4^R)^E = 0, 1 = B_2^R$. For the induction step,

$$\begin{aligned} (B_{2n}^R)^E &= ((B_n^R \cdot (B_n + 1)^R)^E && \text{Fact 1} \\ &= ((B_n^R)^E \cdot ((B_n + 1)^R)^E && \text{Fact 2} \\ &= B_{n/2}^R \cdot (B_{n/2} + 1)^R && \text{Induction hypothesis} \\ &= ((B_{n/2} + 1) \cdot B_{n/2})^R && \text{Fact 1} \\ &= B_n^R \end{aligned}$$

Lemma 15 *The sequence C_n is a fixed point for $\mathcal{M}[n]$.*

Proof By induction. The base case, when $n = 4$, can be verified by inspection.

For the induction step, assume the claim for n, and consider $\mathcal{M}[2n]$. Recall that $C_n = (B_n + 1) \cdot B_n^R$. The input to the first $\mathcal{M}[n]$ network is thus

$$\begin{aligned} (B_n + 1)^E \cdot (B_n^R)^O &= (B_{n/2} + 2) \cdot (B_n^R)^O && \text{Lemma 12} \\ &= (B_{n/2} + 2) \cdot (B_{n/2} + 1)^R && \text{Lemma 13} \\ &= ((B_{n/2} + 1) \cdot B_{n/2}) + 1)^R \\ &= C_n + 1 \end{aligned}$$

By the induction hypothesis, C_n is a fixed point for $\mathcal{M}[n]$, and by Lemma 4, so is $C_n + 1$.

The input to the second $\mathcal{M}[n]$ network is

$$\begin{aligned}(B_n+1)^O \cdot (B_n^R)^E &= (B_{n/2}+1)\cdot (B_n^R)^O && \text{Lemma 11}\\ &= (B_{n/2}+1)\cdot B_{n/2}^R && \text{Lemma 14}\\ &= C_n\end{aligned}$$

By the induction hypothesis, C_n is a fixed point for $\mathcal{M}[n]$.

The input to the last $\mathcal{E}[2n]$ network is the meld $(C_n+1)|C_n$. The input to wire $2i$ is the i-th element of C_n+1, a to wire $2i+1$ is the i-th element of C_n. Since these value differ by one, and the balancers are oriented *up* in $\mathcal{E}[2n]$, the sequence passes through unchanged.

Lemma 16 $\mathcal{M}^*(C_n) = B_n$.

Proof By Lemma 15, $\mathcal{M}(C_n) = C_n = (B_{n/2}+1)\cdot B_{n/2}^R$. By Lemma 10

$$\mathcal{M}^*(C_n) = (B_{n/2}+1)\cdot (B_{n/2}^R)^R = (B_{n/2}+1)\cdot B_{n/2} = B_n.$$

Theorem 2 *The* BITONIC$[n]$ *network has an initial state* $\mathcal{B}[n]$ *for which* B_n *is a* $(\log n)$*-smooth fixed point sequence.*

Proof By induction on n. For the base case, $\mathcal{B}[2]$ is a single balancer oriented *up*.

For the induction hypothesis, assume we have an initial state $\mathcal{B}[n]$ for which B_n is a fixed-point sequence. Construct $\mathcal{B}[2n]$ as follows. Split the BITONIC$[n]$ network into three components: the two parallel BITONIC$[n/2]$ networks, and the final MERGER$[n]$ network.

Initialize the BITONIC$[n]$ network on wires $0,\ldots,w-1$ to $\mathcal{B}[n]$, the other to $\mathcal{B}^*[n]$, and the MERGER$[2n]$ subnetwork to $\mathcal{M}^*[2n]$.

On input $B_{2n} = (B_n+1)\cdot B_n$, the subsequence B_n+1 sent through $\mathcal{B}[n]$ yields B_n+1 (induction hypothesis), and the subsequence B_n sent through $\mathcal{B}^*[n]$ yields B_n^R (induction hypothesis and Lemma 10). The two BITONIC$[n]$ networks thus carry B_{2n} to C_{2n}. By Lemma 16, $\mathcal{M}^*[2n](C_{2n}) = B_{2n}$, so B_{2n} is a fixed-point sequence for $\mathcal{B}[2n]$.

Corollary 1 *It is possible for a* BITONIC$[n]$ *network started in an arbitrary state to yield a sequence which is not* k*-smooth for any* $k < \log(n)$.

4 The periodic counting network

The PERIODIC$[n]$ counting network [2] is isomorphic to the *Periodic* sorting network of Dowd, Perl, Rudolph and Saks [7]. At its heart is a component BLOCK$[n]$ network, defined inductively as follows. The BLOCK$[2]$ network is a single balancer. The BLOCK$[2n]$ network is constructed as follows. Given a sequence X, represent each index (subscript) as a binary string. The *A-cochain* of X, denoted X^A, is the subsequence whose indexes have low-order bits 00 or 11. For example, the *A-cochain* of the sequence $x_0,\ldots,x_7$ is x_0, x_3, x_4, x_7. The *B-cochain* x^B is the subsequence whose low-order bits are 01 and 10.

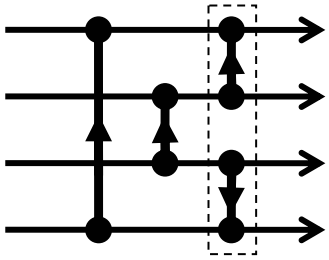

Fig. 8 The $\mathcal{H}[4]$ initial state

The input sequence X is fed into two parallel BLOCK$[n]$ networks, which we will call the *A-block* and the *B-block*. X^A goes to the A-block, and X^B to the B-block. Their output sequences, call them Y^A and Y^B, are fed into an EVENODD$[2n]$ network. The PERIODIC$[n]$ network is just $\log n$ BLOCK$[n]$ networks in series. In this paper, however, we focus on a single BLOCK$[n]$ network. Our upper and lower bounds for the smoothness of the BLOCK$[n]$ network directly carry over to the PERIODIC$[n]$ network. Figure 8 shows a BLOCK$[4]$ network (ignore the arrowheads for now) and Fig. 9 shows a BLOCK$[8]$ network. Here, one component BLOCK$[4]$ is shown in black, the other in gray, and the final EVENODD$[8]$ is highlighted by a dotted box.

4.1 Upper bound

Theorem 3 *The output of a* BLOCK$[n]$ *network initialized in any state is* $(\log n)$*-smooth.*

Proof We argue by induction on n. When n is 2, the network is a single balancer, so the output is 1-smooth.

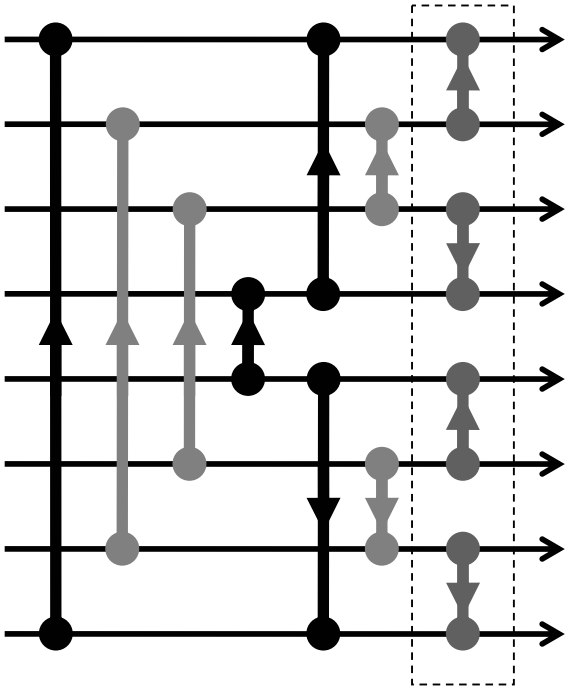

Fig. 9 The $\mathcal{H}[8]$ initial state

In the BLOCK[$2n$] network, the output sequences Y^A and Y^B from the two component BLOCK[n] networks are each $\log n$-smooth by the induction hypothesis, and the final EVENODD[$2n$] layer is a matching for Y^A and Y^B. By Lemma 1, the output is $(\log n + 1)$-smooth.

Corollary 2 *The output of a* PERIODIC[n] *counting network initialized in any state is* $(\log n)$*-smooth.*

Corollary 3 *There exists a balancing network of width n and depth* $\log n$ *whose output is* $(\log n)$*-smooth no matter how the network is initialized.*

4.2 Lower bound

Define the initial state $\mathcal{H}[2]$ to be a balancer oriented *up*. Define the initial state $\mathcal{H}[2n]$ as follows:

- Initialize both the A-block and the B-block to $\mathcal{H}[n]$.
- In the final EVENODD[$2n$] layer, orient balancers between wires $4i$ and $4i + 1$ *up* (for $0 \leq n/4$), and the rest *down*.

Let H_2 be the sequence $1, 0$, and H_{2n} the unique sequence defined by

$$H^A_{2n} = H_n + 1$$
$$H^B_{2n} = H_n$$

Notice that H_n is $(\log n)$-smooth, but not k-smooth for any $k < \log n$. Figure 8 shows balancer orientations for $\mathcal{H}[4]$, and Fig. 9 for $\mathcal{H}[8]$.

Theorem 4 *The* BLOCK[n] *network has an initial state* $\mathcal{H}[n]$ *for which* H_n *is a fixed-point sequence which is not* k*-smooth for any* $k < \log n$.

Proof We argue by induction. For the base case, $\mathcal{H}[2]$ is single balancer oriented *up*.

For the induction hypothesis, assume we have an initial state $\mathcal{H}[n]$ for which H_n is a fixed-point sequence. Initialize the BLOCK[$2n$] network so the two component BLOCK[n] networks have initial states $\mathcal{H}[n]$. On input H_{2n} one component has input $H^A_{2n} = H_n + 1$, which is a fixed-point sequence, and the other has input $H^B_{2n} = H_n$, which is also a fixed-point sequence. The balancer linking wires $2i$ and $2i + 1$ carries the i-th element of $H_n + 1$ on one wire, and the i-the element of H_n on the other. These values differ by one, and the balancer is oriented toward the larger value, so the sequence H_{2n} passes through the final EVENODD[$2n$] layer unchanged.

Corollary 4 *It is possible for a* PERIODIC[n] *network started in an arbitrary state to yield a sequence which is not* k*-smooth for any* $k < \log(n)$.

5 Self-stabilization of balancing networks

What if $(\log n)$-smoothness in the face of failures is not good enough? For example, in an application such as distributed counting, we may need the additional "step property" that correctly initialized counting networks possess. A balancing network is said to possess the *step property* if in every quiescent state, the sequence of the numbers of tokens exiting output wires $1, 2, \ldots n$ possesses the step property (the step property for a sequence is defined in Sect. 2). If such stronger properties are required, then we need a way for the network to recover from faulty states.

In this section, we show how to make simple modifications to balancing networks to ensure that if any such network is reset to an arbitrary illegal state, then it will eventually return to a legal state (formal definitions of legal and illegal states follow). We introduce additional state to the network, and additional actions (which can be "piggy-backed" on tokens), together with simple responses to such actions when it is sent. We call such additional state and actions *self-stabilizing* actions.

We guarantee the following

- If the network is in a legal state, then these actions never occur.
- If the network is in an illegal state, then these actions eventually bring it back to a legal state.

The main idea is as follows. We show how the predicate asserting that the network's state is legal can be written as the conjunction of many simpler predicates, each of which makes *local* assertions about the state of each balancer and wire. As a result, it is enough to stabilize local states involving individual balancers and wires.

Each component periodically checks whether its state is legal. If not, it corrects itself. Correcting one network component might disturb the neighboring component, but we show that such disturbances will flow down the network, from lower to higher depth only.

We now describe the model and precisely define the self-stabilization problem.

5.1 Model

The balancing network is an asynchronous message-passing system, where each balancer is a processor. (A single physical node may possibly implement multiple balancers.) Wires are FIFO network links.

When reasoning about self-stabilization, we assume that balancer states are subject to faults, as are the number of tokens on any given wire. By contrast, we assume that links between nodes are fixed, and so we focus on stabilizing balancers' states, and not on their interconnection. Our self-stabilizing algorithm can be layered on top of another algorithm which stabilizes the interconnection.

For each balancer b we add the following counters, each counter corresponding to one of the wires incident at b:

- $n_i^u(b)$ is the number of tokens that entered b on the upper input wire
- $n_i^\ell(b)$ is the number of tokens that entered b on the lower input wire
- $n_o^u(b)$ is the number of tokens that exited b on the upper output wire
- $n_o^\ell(b)$ is the number of tokens that exited b on the lower output wire

For now, we treat these counters as unbounded. Later, we show how to bound them.

5.2 Legal network states

Informally, a *legal state* of the balancing network is a state that can be reached during normal execution starting from the standard initial state.

More formally, a *transition* occurs when a token residing on a balancer's input wire passes through that balancer, toggles the balancer's state, and emerges on one of the balancer's output wires. The execution of a balancing network is a sequence of transitions, starting from an initial state where all tokens are on network input wires, and ending when all tokens are on network output wires. We say that a transition *occurs at time t* to mean it is the t-th transition starting from the initial network state.

Definition 1 A *standard initial state* of the network is one where all balancers are in the *up* state, all tokens are on network input wires, and every counter at every balancer is zero.

Definition 2 A *legal state* of the network is one that can be reached from a standard initial state by a finite sequence of transitions. Any other state is *illegal*.

Notice that being legal is a global property of the network, since it concerns the states of all balancers and wires simultaneously. This characterization, while natural, is hard to use directly, since no balancer can directly observe the global system state. Instead, we rely on each individual component to check whether its own state is compatible with its neighbor's.

Define $n_i(b) = n_i^\ell(b) + n_i^u(b)$, and $n_o(b) = n_o^\ell(b) + n_o^u(b)$.

Definition 3 A balancer b is *legal* if

1. The number of tokens that have entered b equals the number of tokens that have exited:
$$n_i(b) = n_o(b)$$
2. The tokens that have exited b are balanced on the output wires.
$$n_o^u(b) = \begin{cases} n_o^\ell(b) & \text{if } n_o(b) \text{ is even} \\ n_o^\ell(b) + 1 & \text{if } n_o(b) \text{ is odd} \end{cases}$$
3. The balancer state is *up* if $n_o(b)$ is even, and *down* otherwise.

Consider a wire w directed from balancer b to balancer c

Definition 4 Wire w is *legal* if the counter corresponding to w at balancer b equals the number of tokens in transit on w plus the counter corresponding to w at balancer c.

5.3 Global legality equivalent to local legality

We now state and prove a key theorem, that the predicate that captures whether the global network state is legal can be expressed as the conjunction of local predicates, one for each wire and each balancer.

Theorem 5 *A balancing network is in a* legal *state if and only if every balancer and every wire is in a legal state.*

We prove this theorem in two parts, Lemmas 17 and 18. First, we show that if the balancing network is in a legal state, then every balancer and wire is in a legal state.

Lemma 17 *If a balancing network is in a legal state, then all balancers and wires are in legal states.*

Proof Let S denote a legal state of the balancing network. We will prove this by induction on the number of transitions required to reach S starting from the standard initial state.

For the base case, observe that in any standard initial state, all balancers and wires are in legal states. Each transition is a token passing through a balancer, affecting the balancer's input wire, its output wire, and the balancer itself. We will show that each transition preserves the legality of each component.

- The input wire contains one token less, but the balancer's input counter for that wire is incremented to compensate.
- The balancer's counters for the input and output wire are both incremented.
- The output wire contains one token more, but the balancer's output counter for that wire is incremented to compensate.

A single transition thus preserves legality for each affected component, and the inductive argument shows that all balancers and wires remain in legal states after any sequence of transitions.

We now prove the other direction, that local legality implies global legality.

Lemma 18 *If all the balancers and wires of the network are in legal states, then the global state is legal.*

Proof We show that a network state where every balancer and every wire is legal can be reached from the standard initial state by a sequence of transitions.

A balancer is defined to be *fresh* if all its counters are zero, and it is oriented *up*. Notice that in any legal state, a fresh balancer's output wires are empty. Recall that each

balancer can be assigned a unique *layer*, which is the length of the longest path to that balancer from an input wire.

The *potential* of a network state is defined to be a pair, (ℓ, c), where ℓ is the maximal layer that contains non-fresh balancers, and c is the number of non-fresh balancers. By convention, a network in a standard initial state has potential $(0, 0)$. Potentials are ordered lexicographically, first by level, then by count.

Consider any global state S where each wire and balancer is in a legal state. We will push some tokens "upstream" (to a lower numbered level) to reach a state S' such that S can be reached from S' by a sequence of transitions, but S' has lower potential than S. If we systematically apply this process, we will arrive at state with minimal potential, which must be a standard initial state.

Given S, we define S' as follows. Pick a non-fresh balancer b whose layer is maximal among non-fresh balancers, and let $n_i^\ell(b)$ and $n_i^u(b)$ denote the values of b's counters in S. State S' is the same as S except that

- balancer b is fresh,
- the lower input wire to b has $n_i^\ell(b)$ additional tokens in transit, and
- the upper input wire to b has $n_i^u(b)$ additional tokens in transit.

Note that b and its input wires remain legal in S', a sequence of transitions will take the network from S' to S, and S' has lower potential than S. By repeating this transformation, we see that S can be reached from a standard initial state by a sequence of transitions.

5.4 Self-stabilization algorithm

We have established that if every network element enters a legal state, then the network as a whole enters a legal global state. Next, we describe the local stabilization algorithm in terms of actions that occur along with regular balancing network transitions.

We assume that each balancer is in a legal state, since its legality can be checked locally. Since the program at the balancer is incorruptible, it can ensure that every operation that changes the balancer's state always leaves the balancer in a legal state.

We also assume that every input wire is in a legal state. Since each input wire has only one counter associated with it, which is the number of tokens that have been input on that wire, this is always assumed to be correct. Thus, the input layer of the network needs no stabilization.

We introduce two new transitions. Consider balancers b and c linked by a wire w directed from b to c.

- In the *checking* transition, b sends a *correction token* to c containing b's counter value for that wire. A checking transition is periodically initiated for each wire w, or can be piggy-backed on a regular token traversing the wire.
- In the corresponding *correcting* transition, c receives that correction token, and compares the token's value v with c's counter value for that wire. If they do not match, c resets that counter to v, and updates its two output counters to make c's balancer state legal again. If c changes its counter values, we say c is *disturbed*.

Lemma 19 *If w is in a legal state when b performs a checking transition, then c will not be disturbed when it performs the corresponding correcting transition.*

Proof At the time of the checking transition, let c_b be the value of b's counter for w, c_w the number of tokens on w, and c_c the the value of c's counter for w. Because w is legal, $c_b = c_w + c_c$. Because w is a FIFO link, c will receive the correction token containing c_b only after it has received all c_w tokens on the wire w, at which time its counter value will be $c_c + c_w$, which equals c_b.

Conversely, if c is disturbed by b's correction token, then w was illegal when b performed its checking transition.

Define a wire's *layer* to be the layer of the balancer to which it leads (network input wires have layer zero).

Lemma 20 *If every wire at layer ℓ and lower is in a legal state, then henceforth no balancer at layer ℓ or lower will become disturbed (unless further faults occur).*

Proof By contradiction. Assume that every wire at layer ℓ and lower is in a legal state at time zero. Consider the first subsequent transition in which a balancer at layer ℓ or lower becomes disturbed. This clearly cannot be an input balancer. Suppose wire w is directed from balancer b to balancer c, where c is disturbed by a correcting transition. By Lemma 19, w was in an illegal state at the time b underwent the checking transition. Since w was legal at time zero, it must have changed from legal to illegal at some later time. But the only transition that can make w change from legal to illegal is when b becomes disturbed. We conclude that b must have been disturbed at some time after time zero, but before c became disturbed, contradicting the hypothesis that c was the first balancer to be disturbed.

Lemma 21 *Suppose wire w was directed from balancers b to c. If b is undisturbed in the interval between its checking transition and c's corresponding correcting transition, then w is legal immediately after the correcting transition.*

Proof Let c_0 be the value of b's counter for w at the time of the checking transition. At time following the checking transition but before the correcting transition, let c_w be the number of tokens that have entered w since the checking transition, and c_1 the value of b's counter for w. As long as b is not disturbed, all transitions preserve the invariant $c_1 = c_w + c_0$. When the correcting transition occurs, c's counter for w will be reset to c_0, making w's state legal.

Theorem 6 *If every balancer eventually undergoes a checking transition on each of its output wires, then the network eventually enters a legal state.*

Proof By induction on the number of layers in the network. The claim is immediate for a single-layer network. For the base case, consider a two-layer network. Let b be a balancer in the first layer linked by wire w to balancer c in the second layer. By hypothesis, b will eventually send a correction token to c. Balancer b will never be disturbed, because there are no balancers to send it correction tokens. Lemma 21 states that when c undergoes the corresponding correcting transition, w will be legal. Because b will never be disturbed, w will henceforth remain legal.

For the induction step, assume the result for networks of depth less than d. By the induction hypothesis, all wires at layers less than d will eventually enter legal states. From that time on, Lemma 20 states that no balancer at layer $d-1$ will ever become disturbed. The rest of the argument proceeds as in the base step.

Time to stabilization: One parallel stabilization step is the time until every linked pair of balancers has executed a checking and corresponding correcting transition. The network stabilizes in d parallel stabilization steps, where d is the network depth. Checking steps can be initiated by a timeout, and the frequency of initiation of the stabilization step can be controlled at each balancer, and once the step is initiated it completes in time equal to the latency of the wire.

Lazy stabilization: Instead of proactively and periodically sending out correction tokens, we could piggy-back the correction tokens on regular tokens (for example, the counting tokens). The time to stabilization will now depend on the rate at which tokens enter the network.

Transient behavior: The stabilization guarantees only that the network will eventually behave in a legal way. For example, if the network is a k-smoothing network, then network output could fail to be k-smooth while stabilization is in progress.

5.5 Extra space needed for self-stabilization

We now show how to bound the sizes of the counters used for self-stabilization. Suppose the width of the network is n, i.e. the maximum number of balancers in any layer of the network is n. Let d denote the depth of the network.

Theorem 7 *The execution of the balancing network would not change if we replaced the counters corresponding to a wire connecting a balancer at depth j to a balancer at depth $j+1$ by counters modulo 2^{d-j}, where d denotes the depth of the network.*

Proof Proof by reverse induction on j, starting with $j = (d-1)$, the last layer. A balancer in the last layer chooses its next output wire based only on the parity of the number of tokens that have entered the balancer. That parity, in turn, is determined by the parities of the numbers that have entered on the upper and lower input wires. A balancer on layer $d-1$ therefore needs two one-bit input counters.

For the induction step, assume balancers at layer j require input counters modulo 2^{d-j}. Balancers at layer $j-1$ therefore require output counters of the same size. To keep track of both values, however, that balancer needs input counters modulo 2^{d-j+1}.

Space complexity: Since the counters at layer k are modulo 2^{d-k}, all the counters at layer k together require at most $2n(d-k)$ bits of space. Summing over all layers, the additional space required for stabilization is no more than

$$\sum_{i=1}^{d} 2ni = nd(d+1).$$

6 Conclusions

We have investigated smoothing properties of popular balancing networks when they start from adversarially chosen initial states. We found that the Bitonic and Periodic networks of width n are always $(\log n)$-smooth, irrespective of the state they start in. Further, for both the above networks there exist starting states and input sequences such that the output sequence is no smoother than $\log n$. This implies that if only approximate smoothness is required from the network, then nothing need be done to recover from faults. In more recent work [8], we have shown that if the Block, Periodic or Bitonic networks are started from randomly chosen starting states, then the output smoothness is even better, and is $O(\sqrt{\log n})$.

In situations where the network must provide stronger properties than $\log n$-smoothness, such as the step property, we give a simple self-stabilizing transformation that ensures that any balancing network in an illegal state will converge to a legal state.

References

1. Aiello, W., Venkatesan, R., Yung, M.: Coins, weights and contention in balancing networks. In: Proceedings of the ACM Symposium on Principles of Distributed Computing, pp. 193–205 (August 1994)
2. Aspnes, J., Herlihy, M., Shavit, N.: Counting networks. J. ACM, **41**(5), 1020–1048 (1994)
3. Batcher, K.E.: Sorting networks and their applications. In: Proceedings of the AFIPS Spring Joint Computer Conference. vol 32, pp. 338–334 (1968)
4. Busch, C., Herlihy, M.: A survey on counting networks. In Proceedings of Workshop on Distributed Data and Structures (WDAS). (March 1998)
5. Dijkstra, E.W.: Self stabilizing systems in spite of distributed control. Communications of the ACM, **17**, 643–644 (1974)
6. Dolev, S.: Self-Stabilization. The MIT Press. (2000)
7. Dowd, M., Perl, Y., Rudolph, L., Saks, M.: The periodic balanced sorting network. J. ACM, **36**(4), 738–757 (October 1989)

8. Herlihy, M., Tirthapura, S.: Randomized smoothing networks. In Proceedings of the International Parallel and Distributed Processing Symposium. (April 2004)
9. Reidel, M., Bruck, J.: Tolerating faults in counting networks. In IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems. (April 1999)
10. Schneider, M.: Self-stabilization. ACM Computing Surveys, **25**, 45–67 (1993)