

Self-Stabilizing Smoothing and Counting

Maurice Herlihy
Computer Science Department
Brown University
mph@cs.brown.edu

Srikanta Tirthapura
Dept. of Electrical and Computer Engg.
Iowa State University
snt@iastate.edu

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.

Second, we show that the same networks can be made eventually 1-smooth by “piggy-backing” a small amount of additional information on messages when (and only when) trouble is detected.

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 active tokens.

Smoothing networks are well-suited for load-balancing applications where tokens represent requests for service. Clients send tokens to arbitrary input wires, and those 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 work under these new 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, any such network with w input and output wires, when started in any of its $O(2^w)$ possible initial states, will produce outputs such that all the number of tokens emerging on output wires lie within $\log(w)$ of one another. This bound is tight.

The second contribution of this paper is to show that counting networks can be made *self-stabilizing*, so that even if the network is started in an arbitrary state, it eventually converges to a 1-smooth state. The information required for self-stabilization can be piggy-backed on existing messages.

2. Model

A *balancer* is an asynchronous switch with two input wires and two output wires, 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 token leaves on wire 0, and the balancer becomes oriented *down*. If, however, the balancer is oriented *down*, then the next token leaves on 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.

A *balancing network* is an acyclic network of balancers where output wires of some balancers are linked to input wires of others. The *standard* initial state of a balancing network is to have all balancers oriented *up*.

The network's *input wires* are those input wires not linked to the output of any balancer, and similarly for the

network's *output wires*. In this paper, 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 balancing network is a *k-smoothing network* if, starting from its standard initial state, the overall distribution of output tokens across the output wires is *k-smooth*: exiting tokens are divided among the output wires in such a way that no wire has more than *k* tokens more than any other (a formal definition is given below).

A *counting network* [3] is a 1-smoothing network with certain additional properties that are important in other contexts, but do not concern us here.

This paper addresses some new questions about smoothing networks. The literature on balancing and counting networks has always assumed that networks are initialized in their standard starting states. We 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 simple. 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 will show that well-known counting networks such as the bitonic and the periodic networks are remarkably smooth even when not initialized to the standard initial state: a network of width *w* is $\log(w)$ -smooth when started in any of its $\Omega(2^w)$ possible initial states. Moreover, this bound is tight.

We then show that we can restore the smoothing properties of any *k-smooth* network, started in an arbitrary state, back to *k-smooth*, by "piggy-backing" a small amount of additional information on tokens. (Naturally, this technique is interesting mostly for 1-smoothing networks). The result is a simple protocol for making *k-smoothing* networks *self-stabilizing*: once the additional information has propagated through the *k-smoothing* network, it resumes acting like a *k-smoothing* network that was started in the correct initial state. We analyze the time and (additional) space required for stabilization. The time to stabilization is proportional to the depth of the counting network.

- If the network is in a legal state (we will later define "legal state" precisely), then self-stabilizing actions do not occur.
- If the network is not in a legal state, then the self-stabilizing actions eventually bring the network back to a legal state.

The key advantage of self-stabilization is that no external action is needed to initiate the recovery from faults. In addition, the stabilizing actions can take place in parallel with the normal execution of the protocol.

The rest of the paper is organized as follows. Section 3 proves some general smoothing properties for balancing networks. Sections 4 and 5 analyze the smoothing properties of the Bitonic and the Periodic counting networks respectively. In Section 6 we show how to make counting networks self-stabilizing.

3. Balancing Networks

We denote sequences of natural numbers in upper case, and elements of a sequence in lower case. For example, a sequence $X = x_0, \dots, x_{w-1}$ has length (or width) $|X| = w$. Sequence X is *k-smooth* if $|x_i - x_j| \leq k$, for any $0 \leq i, j < w$. The elements of a *k-smooth* sequence take values in the range $a, a + 1, \dots, a + k$ for some *a*.

If X and Y are sequences of length *w*, $X \cdot Y$ denotes their concatenation: $x_0, \dots, x_{w-1}, y_0, \dots, y_{w-1}$. If X is a sequence, $X + c$ denotes the sequence $x_0 + c, x_1 + c, \dots$.

Let X and Y be *k-smooth* sequences of length *n*. 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.

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_l and z_{l+1} . If the balancer is oriented up, then

$$\begin{aligned} z_l &= \left\lfloor \frac{x_i + y_j}{2} \right\rfloor \\ z_{l+1} &= \left\lceil \frac{x_i + y_j}{2} \right\rceil \end{aligned}$$

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: A balancer can never increase the difference between two sequence elements. ■

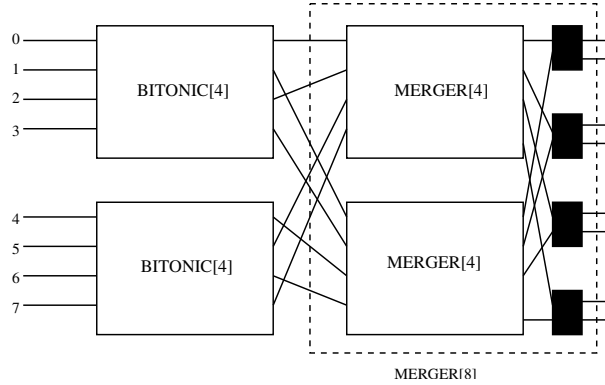


Figure 1. Recursive Structure of a BITONIC[8] Counting Network.

4. The Bitonic Counting Network

The BITONIC[w] counting network [3] is isomorphic to the *Bitonic* sorting network of Batcher [4]. (Henceforth, any network width w will be a power of 2.) This network has a simple inductive structure. The BITONIC[2] network is a single balancer. As illustrated in Figure 1, the BITONIC[$2w$] network is constructed by feeding the $2w$ input wires into two parallel BITONIC[w] networks, and feeding their outputs into a MERGER[$2w$] network.

The MERGER[$2w$] balancing network takes two input sequences X and Y , each of length w , and produces an output sequence Z of length $2w$. The MERGER[w] network is also defined inductively. The MERGER[2] network is a single balancer. We construct the MERGER[$2w$] network from two MERGER[w] networks and w balancers. Let X^E denote the even subsequence x_0, x_2, \dots, x_{w-2} of X , and X^O denote the odd subsequence x_1, x_3, \dots, x_{w-1} . Similarly define Y^E and Y^O .

The input to the first MERGER[w] network is the sequence $X^E \cdot Y^O$. Call that output sequence U . Symmetrically, the input to the second MERGER[w] network is the sequence $X^O \cdot Y^E$. Call that output sequence V . The final layer of the network combines U and V by sending each pair of wires u_i and v_i into a balancer whose outputs yield Z_{2i} and Z_{2i+1} .

4.1. Upper Bound

Let X and Y denote the two input sequences into MERGER[$2w$], which are also the output sequences of the two BITONIC[w] networks.

Lemma 3 *The first layer of the MERGER[$2w$] network is a matching between X and Y .*

Proof: We argue inductively. The base case, when $w = 2$,

is trivial. As the induction hypothesis, assume that the first layer of MERGER[w] is a matching.

The first layer of MERGER[$2w$] is just the first layer of both MERGER[w] 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 . ■

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

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

In the BITONIC[w] network, the two output sequences X and Y from the two component BITONIC[$w/2$] networks are each $(\log w - 1)$ -smooth by the induction hypothesis. These sequences are fed to a MERGER[w] network, which is a matching for X and Y by Lemma 3. Lemma 1 and Lemma 2 together imply that the output sequence Z is $\log w$ -smooth. ■

4.2. Lower Bound

We now show that Theorem 1 is tight. We will display an input sequence S_k and an initial state $\mathcal{B}[w]$ for a BITONIC[w] network with the property that the output sequence B_w is $\log(w)$ -smooth.

A *fixed-point* sequence for a balancing network B is an initial state \mathcal{B} of the network, and a sequence S with the property that feeding S into \mathcal{B} yields S as an output sequence.

Lemma 4 *If S is a fixed-point sequence for a balancing network B , then so are $S + c$, for any constant c , and S^R .*

If $\mathcal{B}[w]$ is any initial state of the BITONIC[w] network, define $\mathcal{B}^*[w]$ to be the initial state constructed by reversing the orientation of every balancer in the network's MERGER[w] component.

Lemma 5 *If input sequence X in $\mathcal{B}[w]$ yields output sequence Y , then input sequence X in $\mathcal{B}^*[w]$ yields output sequence Y^R .*

Proof: (Sketch) The first layer of the MERGER[w] network links wire i to wire $n - i - 1$. If X yields sequence M after the first layer of the MERGER[w] network from initial state $\mathcal{B}[w]$, then X yields sequence M^R after the first layer of the MERGER[w] network from initial state $\mathcal{B}^*[w]$, and the sequences at each successive layer remain reversed. ■

Let $\mathcal{M}[w]$ is any initial state of the MERGER[w] network. Define $\mathcal{M}^*[w]$ to be the initial state constructed by reversing the orientation of every balancer linking wires $w/2, \dots, w - 1$.

Lemma 6 *If input sequence X to BITONIC[w] in $\mathcal{B}[w]$ yields output sequence $Y \cdot Z$, where Y and Z have length $w/2$, then input sequence X to BITONIC[w] in $\mathcal{M}^*[w]$ yields output sequence $Y \cdot Z^R$.*

Proof: (Sketch) We argue by induction on w . In the base case, where $w = 4$, $\mathcal{M}^*[4]$ reverses the lowest balancer on the last layer, reversing the order of outputs 2 and 3, while leaving outputs 0 and 1 unchanged.

Assume the claim for $\mathcal{M}[w]$. Recall that a MERGER[$2w$] network consists of two MERGER[k] networks with a final layer linking the i -th wires of each component. From input sequence X , in state $\mathcal{M}[2w]$, let $U_0 \cdot V_0$ and $U_1 \cdot V_1$ be the component networks' respective output sequences, all of length w . In state $\mathcal{M}^*[2w]$, the component networks are both in state $\mathcal{M}^*[w]$, so by the induction hypothesis, input X yields output sequences $U_0 \cdot V_0^R$ and $U_1 \cdot V_1^R$, where the reversed elements are on wires w and higher. The balancers in the final layer of wires connecting the reversed elements are themselves reversed in \mathcal{B}^* , so the final output sequence on those wires is also reversed. ■

We define two sequences of interest:

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

Notice that B_w is $\log(w)$ -smooth.

Theorem 2 *The BITONIC[w] network has an initial state $\mathcal{B}[w]$ for which B_w is a fixed point.*

Proof: By induction on w . The claim is immediate for $w = 2$.

Assume we have an initial state $\mathcal{B}[w]$ for which B_w is a fixed point. To construct $\mathcal{B}[2w]$, initialize the BITONIC[w] network on wires $0, \dots, w - 1$ to $\mathcal{M}[w]$, and the other

to $\mathcal{M}^*[w]$. On input $B_{2w} = (B_w + 1) \cdot B_w$, the subsequence $B_w + 1$ sent through $\mathcal{M}[w]$ yields $B_w + 1$, and the subsequence B_w sent through $\mathcal{M}^*[w]$ yields B_w^R . The two BITONIC[w] networks thus carry B_{2w} to C_{2w} .

Let $\mathcal{M}[2w]$ be the standard initial state for the MERGER[$2w$] network (where all balancers point up). It is easy to check that C_{2w} is a fixed-point sequence for $\mathcal{M}[2w]$. We construct $\mathcal{B}[2w]$, however, by initializing the MERGER[$2w$] network to $\mathcal{M}^*[2w]$, which carries C_{2w} to $(B_w + 1) \cdot (B_w^R)^R = B_{2w}$. ■

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

5. The Periodic Counting Network

The PERIODIC[w] counting network [3] is isomorphic to the *Periodic* sorting network of Dowd, Perl, Rudolph and Saks [8]. We first define a component BLOCK[w] network, illustrated in Figure 2. The BLOCK[2] network is a single balancer. The BLOCK[$2w$] network is constructed as follows. Given a sequence X , it is convenient to represent each index (subscript) as a binary string. The *A-cochain* of X , denoted X^A , is the subsequence whose indices have the two low-order bits 00 or 11. For example, the *A-cochain* of the sequence x_0, \dots, 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.

The input sequence W is fed into two parallel BLOCK[w] networks, where W^A goes to one network, and W^B to the other. Call the output sequences X and Y . A final layer joins each x_i and y_i by a balancer, yielding outputs z_{2i} and z_{2i+1} . The PERIODIC[w] network is just $\log(w)$ BLOCK[w] networks in series.

5.1. Upper Bound

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

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

In the BLOCK[$2w$] network, the two output sequences X and Y from the two component BLOCK[w] networks are each $\log w$ -smooth by the induction hypothesis, and the final layer of balancers is a matching for X and Y . Lemma 1 implies that the output sequence Z is $(\log w + 1)$ -smooth. ■

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

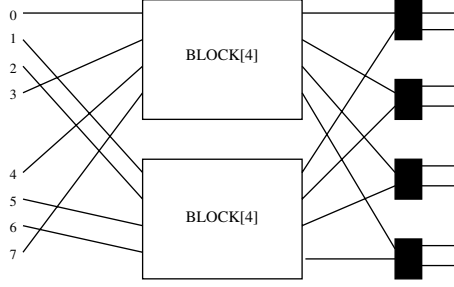


Figure 2. Recursive Structure of a BLOCK[8] Network.

5.2. Lower Bound

Define H_2 to be the sequence 1, 0, and H_{2w} to be the unique sequence satisfying

$$\begin{aligned} H_{2w}^A &= H_w + 1 \\ H_{2w}^B &= H_w \end{aligned}$$

Notice that H_w is $\log(w)$ -smooth, but not k -smooth for any $k < \log(w)$.

Theorem 4 H_w is a fixed-point sequence for BLOCK[w].

Proof: We argue by induction. For $w = 2$, the claim is immediate.

Assume the claim for w , and let $\mathcal{H}[w]$ be the initial network state for w . Initialize the BLOCK[$2w$] network so the two component BLOCK[w] networks have initial states $\mathcal{H}[w]$. On input H_{2w} one component has input $H_{2w}^A = H_w + 1$, which is a fixed-point sequence, and the other has input $H_{2w}^B = H_w$, which is also a fixed-point sequence. Each balancer in the final row joins one element of $H_w + 1$ and H_w . Orient each such balancer “toward” the larger value. ■

Corollary 3 H_w is a fixed-point sequence for PERIODIC[w].

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

6. Self-stabilization of Counting Networks

What if $\log(w)$ -smoothness in the face of failures is not good enough? In this section, we show how to make simple modifications to smoothing networks to ensure that if any such network is reset to an arbitrary illegal state, then it will eventually return to a legal state. We introduce a small amount of additional information (which can be “piggy-backed” on tokens), together with simple responses to such

information when it is sent. We call such additional information (and reactions) *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, then 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.

6.1. Model

The smoothing 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.

6.2. Legal Network States

Informally, a *legal state* of the counting network is a state that can be reached during normal execution from the standard initial state. More formally, a *transition* occurs when a token passes through a balancer, changing the balancer's state, and the state of the input and output wires. The execution of a counting network is a sequence of transitions, starting from an initial state.

Definition 1 *The standard initial state of the network is one where all balancers are pointing up, there are no tokens in transit, 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 defined as a global property of a 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 the following conditions are satisfied:*

1. *The number of tokens that have entered the balancer equals the number of tokens that have exited:*

$$n_i(b) = n_o(b)$$

2. *The tokens that have exited b thus far are balanced on the output wires.*
 - *If $n_o(b)$ is even, then $n_o^u(b) = n_o^\ell(b)$, and the balancer is pointing upwards.*
 - *If $n_o(b)$ is odd, then $n_o^u(b) = n_o^\ell(b) + 1$, and the balancer is pointing downwards.*

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 .*

6.3. Global Legality Equivalent to Local Legality

We now show how the predicate that captures whether the network state is legal can be expressed as the conjunction of local predicates, one for each wire and each balancer.

Theorem 5 *A counting 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.

Global implies Local: First, we show that if the counting network is in a legal state, then every balancer and wire is in a legal state. Let S denote a legal state of the counting network.

By definition, there must be a sequence of transitions starting from the standard initial state and leading to S . In the standard initial state it is easy to verify that all balancers and wires are in a legal state. We will show that each transition will preserve the legality of the local states.

Each transition is a token passing from an input wire of a balancer to an output wire. This affects the state of three elements of the network:

- The input wire to the balancer (on which that token entered)
- The output wire of the balancer (on which the token exited) and
- The balancer itself

It can be verified that the legality of each of these network elements is preserved during a transition. We will verify the case of the input wire here. The input wire has one less token in transit, but the counter at the destination end of the wire has increased by one. Since the wire was in a legal state to begin with, it remains legal.

Local implies Global: We now prove the other direction. If all the balancers and wires of the network are in legal states, then the global state is legal. We show that such a network state can be reached from the standard initial state by a sequence of transitions.

We number the balancers of the network as follows. Let w denote the width of the network. The balancers in the output stage are numbered from 1 (topmost balancer) to w (bottom balancer). The balancers in the stage adjacent to the output stage are numbered from $w + 1$ onwards, and so on. Observe that the numbers increase as we proceed from the output to the input stage. A balancer is defined to be *fresh* if all its counters are zero and it is pointing upward. Similarly, a wire is *fresh* if there are no tokens in transit on the wire.

Consider any global state S where each wire and balancer is in a legal state. We push some tokens "upstream"

to get a state S' such that S can be reached from S' by a sequence of transitions, and S' has one more fresh balancer than S . If we systematically apply this process, we will arrive at a state where every balancer is fresh, which is the standard initial state.

Given S , we define S' as follows. Let b be the lowest numbered balancer which is not fresh. State S' is the same as S except:

- Balancer b is fresh and its output wires are also fresh
- The lower input wire to b has $n_i^\ell(b)$ extra tokens in transit (where $n_i^\ell(b)$ and $n_i^u(b)$ denote the value of the counters in state S , not S')
- The upper input wire to b has $n_i^u(b)$ extra tokens in transit

It can be verified that that S can be reached from S' by a sequence of legal transitions. We omit the details here. By iterating this process, we can increase the number of fresh balancers and wires, until we reach the standard initial state. This shows that the state S can be reached from the standard initial state by a sequence of transitions.

6.4. Self-stabilization Algorithm

We now know that if we stabilize every network element into a legal state, we have stabilized the network into a legal global state. Next, we describe the local stabilization algorithm. Note that these actions would be occurring along with the regular transitions of the counting network.

We can assume that each balancer is always 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.

Action 1: Every balancer b (except for the output balancers) periodically sends out a "correction" token on the upper output wire with data $n_o^u(b)$, and a corresponding correction token on its lower output wire with data $n_o^\ell(b)$. Note that these "correction" tokens do not increment $n_o^u(b)$ and $n_o^\ell(b)$.

Action 2: When a balancer b receives a "correction" token on an input wire (say, the lower wire), with data n

1. It compares n with the counter corresponding to the input wire ($n_i^\ell(b)$ in this case)
2. If the two match, then do nothing.

Otherwise, assign n to $n_i^\ell(b)$, and change the output counters on the balancer accordingly, to maintain the balancer in a legal state. In such a case, we say that the input wire was *enabled*.

If Action 2 changed any (or both) of the output counters on the balancer, then we say that the corresponding output wires were *disturbed*. We have the following important observation about the self-stabilization actions.

Observation 1 *Action 2 on wire r can disturb only those wires which are adjacent to r and are at a greater depth than r .*

6.5. Proof of stabilization

We sketch the proof by a sequence of lemmas.

Lemma 7 *If the system is in a legal state S , then no wire is enabled.*

Proof: From Theorem 5, we know that each wire is in a legal local state in S . It can be verified that a wire in a legal state is never enabled. ■

A transmission of a correction token by the origin balancer (Action 1) of a wire followed by a receipt of the same by the destination balancer of the wire (Action 2) is called a *stabilization step* for the wire. The time interval between the transmission and receipt is called the *duration* of the stabilization step.

Lemma 8 *If a wire was in an illegal state, and the following events occurred:*

- (1) *A stabilization step was completed for the wire, and*
 - (2) *In the duration of the stabilization step, the wire was not disturbed,*
- then the wire will come to a legal state.*

Proof: It can be verified that the stabilization step sets the counters on the wire to the correct values to bring it to a legal state. ■

Let d denote the depth of the counting network. The depth of the balancers range from 0 (the input stage) to $d-1$ (the output stage). The depth of a wire is defined as the depth of the destination end of the wires, and the output wires are defined to be at depth d . Thus, the input wires are at depth 0.

Lemma 9 *For $k = 1 \dots d-1$, if all wires at depth $k-1$ or lesser are in their respective legal states, then each wire in depth k will reach a legal state after it completes a stabilization step. The wire will never be enabled again (unless further faults occur).*

Proof: (Sketch) Proof is by induction on k .

Base case: Consider a wire r at depth 1. From Observation 1, r can only be disturbed by a wire at the input layer, which is not possible. Lemma 8, implies that r will come to a legal state. Furthermore, since r can never be disturbed, it will remain in a legal state. The proof of the inductive case is similar. ■

Fact 1 Eventually, a stabilization step occurs for every wire.

Theorem 6 If it started out in a faulty state, then the network stabilizes after d parallel stabilization steps, where d denotes the depth of the network.

Proof: First note, that the input wires are always in legal states. From Lemma 9 and Fact 1, all wires at depth 1 will have stabilized after each of them has completed a stabilization step (one parallel stabilization step). Arguing inductively, all wires in layer k will have stabilized after k parallel stabilization steps, and the whole network will stabilize in d parallel steps. ■

Time to stabilization: One parallel stabilization step is the time elapsed until every wire in the layer has executed one stabilization step. The network stabilizes in d steps where d is the depth, and “step” is the maximum time elapsed before a stabilization step is initiated and completed for a wire. The stabilization step is usually 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 pro-actively and periodically sending out correction tokens, we could piggyback the correction tokens on regular (counting) tokens. The time to stabilization will now depend on the rate at which tokens are entering the system.

Transient Behavior: The stabilization only guarantees that the network will eventually start behaving like a counting network, and tokens will be evenly balanced on the output wires from then onwards. Till then, however, the step property may not be obeyed, and the tokens might be unevenly distributed across the outputs.

6.6. Extra space needed for self-stabilization

So far, we have been working as if the additional counters that were introduced at the balancers for self-stabilization are unbounded integers. We now show how to work with bounded integers.

Theorem 7 The execution of the counting 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 induction on j , starting with $j = (d - 1)$. Consider a wire connecting a balancer at depth $d - 1$ to a balancer at depth d . For a balancer b at depth d , the only thing that matters to its state is the parity of the number of tokens that entered b . Thus, it suffices to store the input counters to the wires modulo 2, and the output counters for balancers at depth $d - 1$ need to be stored modulo 2.

If we wanted to count the number of output tokens at a balancer modulo 2^k , then it will do to count the number of input tokens modulo 2^{k+1} . Proceeding thus, we find that it suffices to keep track of the counters at the input to the balancer modulo 2^d . ■

Space Complexity: Since the counters at depth k are modulo 2^{d-k} , it takes $w(d - k)$ bits for depth k , if we assume that the width of each stage of the network is w . The total additional space needed for self-stabilization is (in number of bits):

$$\sum_{i=1}^d wi = wd(d + 1)/2$$

Acknowledgments: We thank the reviewers for comments which helped improve the presentation.

References

- [1] E. Aharonson and H. Attiya. Counting networks with arbitrary fan-out. *Distributed Computing*, 8:163–169, 1995.
- [2] W. Aiello, R. Venkatesan, and M. Yung. Coins, weights and contention in balancing networks. In *Proceedings of the annual ACM symposium on Principles of Distributed Computing*, pages 193–205, August 1994.
- [3] J. Aspnes, M. Herlihy, and N. Shavit. Counting networks. *Journal of the ACM*, 41(5):1020–1048, 1994.
- [4] K.atcher. Sorting networks and their applications. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 32, pages 338–334, 1968.
- [5] C. Busch, N. Hardavellas, and M. Mavronicolas. Contention in counting networks. In *Proceedings of the 13th annual ACM symposium on Principles of Distributed Computing*, page 404, August 1994.
- [6] T. Cormen, C. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge MA, 1990.
- [7] E. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the ACM*, 17:643–644, 1974.
- [8] M. Dowd, Y. Perl, L. Rudolph, and M. Saks. The periodic balanced sorting network. *Journal of the ACM*, 36(4):738–757, October 1989.
- [9] M. Riedel and J. Bruck. Tolerating faults in counting networks. Technical report, California Institute of Technology. Electronic Technical Report ETR022.
- [10] M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25:45–67, 1993.