Tolerating Faults in Counting Networks

MARC D. RIEDEL AND JEHOSHUA BRUCK {riedel,bruck}@paradise.caltech.edu
California Institute of Technology

Abstract. Counting networks were proposed by Aspnes, Herlihy and Shavit [4] as a technique for solving multiprocessor coordination problems. We describe a method for tolerating an arbitrary number of faults in counting networks. In our fault model, the following errors can occur dynamically in the counting network data structure: 1) a balancer's state is spuriously altered, 2) a balancer's state can no longer be accessed.

We propose two approaches for tolerating faults. The first is based on a construction for a *fault-tolerant* balancer. We substitute a fault-tolerant balancer for every balancer in a counting network. Thus, we transform a counting network with depth $O(\log^2 n)$, where n is the width, into a k-fault-tolerant counting network with depth $O(k \log^2 n)$.

The second approach is to append a *correction network*, built with fault-tolerant balancers, to a counting network that may experience faults. We present a bound on the error in the output token distribution of counting networks with faulty balancers (a generalization of the error bound for sorting networks with faulty comparators presented by Yao & Yao [21]). Given a token distribution with a bounded error, the correction network produces a token distribution that is smooth, i.e., the number of tokens on each output wire differs by at most one (a weaker condition than the step property). In order to tolerate k faults, the correction network has depth $O(k^2 \log n)$ for a network of width n.

Keywords: counting networks, fault tolerance, concurrent data structures, load balancing, network routing

1. Introduction

Shared counting is the basis for many fundamental multiprocessor coordination algorithms, such as scheduling, load balancing and resource allocation. Such algorithms typically require that processes cooperate to assign consecutive integer values from a given range. The usual approach is to serialize access to a single shared counter value. However, due to high contention, accessing the counter value becomes a sequential bottleneck.

Counting networks were proposed by Aspnes, Herlihy and Shavit [4] as a low-contention data structure for multiprocessor coordination. In abstract terms, counting networks are constructed from two-input, two-output routing devices called balancers, connected by wires. Each balancer receives tokens on its input wires and forwards these alternately to its top and bottom output wires. Processes make increment requests by placing tokens on designated input wires. The network routes input tokens to designated output wires in increasing order modulo n, where n is the width of the network. Counter values are assigned based upon the output wire number. In shared-memory architectures, each balancer is implemented with a boolean variable representing its state, as well as two pointers to successor balancers. Processes shepherd tokens through the network, toggling balancers through atomic *read-modify-write* operations.

Counting networks achieve high throughput by permitting multiple requests for counter values to proceed concurrently. Each request accesses only a small fraction of the balancers, so the contention on each balancer is low. Aspnes et al. give convincing experimental evidence that counting networks have higher throughput than conventional implementations when the load on the network is sufficiently high [4]. Counting networks are nonblocking: if a process experiences failures or delays while shepherding a token, this will not prevent other processes from making progress. However, this model cannot tolerate corrupted data bits in shared memory.

In this paper, we address the issue of adding fault tolerance to counting networks. In our fault model, the following errors can occur dynamically in the counting network data structure: 1) a balancer's state is spuriously altered, 2) a balancer's state can no longer be accessed. With the occurrence of faults, the distribution of tokens at the output of a counting network may no longer satisfy the step property required for counting. We present an upper bound on the error in the output token distribution of counting networks with faulty balancers. This is a generalization of the error bound for sorting networks with faulty comparators presented by Yao & Yao [21].

We propose two approaches for tolerating faults. The first is based on a construction for a *fault-tolerant* balancer. We substitute a fault-tolerant balancer for every balancer in a counting network. Thus, we transform a counting network with depth $O(\log^2 n)$, where n is the width, into a k-fault-tolerant counting network with depth $O(k \log^2 n)$. (All logarithms are base 2.)

The second approach is to append a *correction network*, built with fault-tolerant balancers, to a counting network that may experience faults. Given an output token distribution with a bounded error from the counting network, the correction network produces a token distribution that is smooth, i.e., the number of tokens on each output wire differs by at most one. This is a weaker condition than the step property of counting networks; however, for applications such as load balancing it is sufficient. In order to tolerate k faults, the correction network has depth $O(k^2 \log n)$ for a network of width n. Figure 1 shows a correction network appended to a counting network of width four. The counting network experiences a fault resulting in a token distribution of $3, 2, 1, 1$ on wires 0 through 3. As the reader can verify, the correction network produces a smooth token distribution of $2, 2, 2, 1$.

1.1. Related Work on Counting Networks

Aspnes et al. [4] first introduced the concept of counting networks. They presented two constructions for counting networks of depth $O(\log^2 n)$, where n is the width. Herlihy et al. [9] discussed implementation issues and presented experimental results comparing the performance of counting networks to other synchronization techniques. Klugerman and Plaxton [12] proved the existence of counting networks of depth $O(\log n)$. Busch and Mavronicolas [5][6][14] investigated the combinatorial

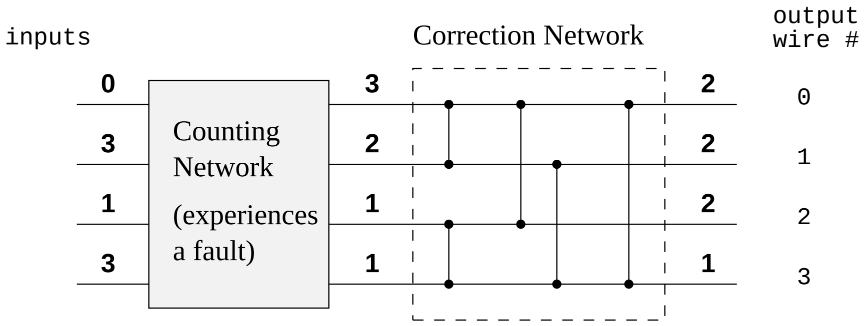

Figure 1. A correction network appended to a counting network experiencing a fault.

structure of balancing networks. Aharonson and Attiya [3], Felton et al. [7] and Hardavellas et al. [8] proposed networks of balancers with fan-out greater than two. Herlihy et al. [10], Lynch et al. [13] and Mavronicolas et al. [15] studied conditions under which counting networks are *linearizable*, i.e., the ordering of values assigned corresponds exactly to the real-time order in which requests are made.

2. Balancers and Counting Networks

Counting networks belong to a class of data structures called *balancing networks*. In abstract terms, a balancing network is a directed acyclic graph whose nodes are asynchronous routing devices called *balancers*, each with in-degree two and out-degree two, and whose edges are communication channels called *wires*. A balancer, shown in Figure 2(a), is a device which receives items called *tokens* on its input wires and forwards these alternately to its top and bottom output wires. It is often convenient to represent a balancer as two dots joined by a vertical bar, as shown in Figure 2(b). In our figures, tokens are labeled according to their order of arrival, but these labels are not used by the network.

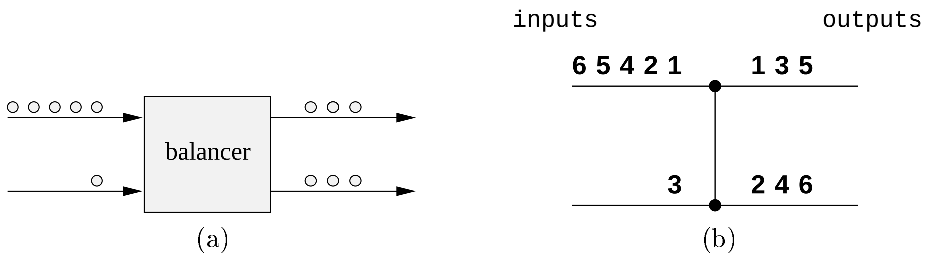

Figure 2. A balancer.

Figure 3 shows an example of a balancing network, the `BITONIC[4]` counting network from [4]. Seven tokens are placed on randomly chosen input wires and

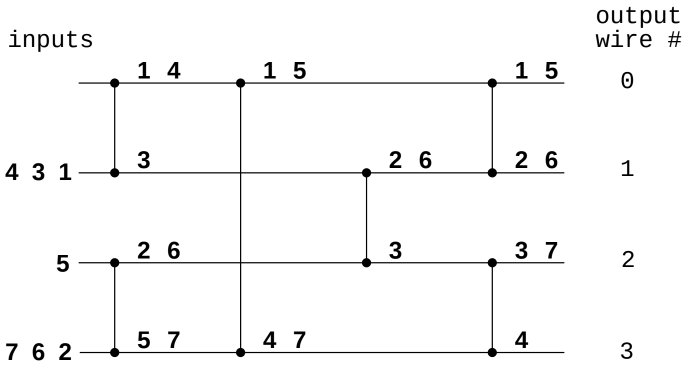

Figure 3. Routing of tokens in a `BITONIC[4]` counting network.

routed to output wires by the balancers. Note that the first token arrives at output wire 0, the second at output wire 1, and so on. In general, the n-th token arrives at output wire n mod 4. This is the defining property of counting networks.

A balancer requires a single bit of memory that describes its *state*. If the state is `up`, it forwards the next token to the top output wire. If the state is `down`, it forwards the next token to the bottom output wire. Denote by x_t and x_b the number of tokens received on a balancer's top and bottom input wires, respectively. Similarly, denote by y_t and y_b the number of tokens forwarded to its top and bottom output wires, respectively.

The safety and liveness properties of a balancer are as follows (see [4]):

1. It never creates tokens, i.e., $x_t + x_b \geq y_t + y_b$.

2. It never swallows tokens, i.e., eventually $x_t + x_b = y_t + y_b$.

3. It balances the number of tokens on its output wires, i.e., when all tokens have been forwarded through

$$y_t = \left\lceil \frac{x_t + x_b}{2} \right\rceil, \; y_b = \left\lfloor \frac{x_t + x_b}{2} \right\rfloor .$$

For a balancing network, define the *depth* of a wire to be zero for a network input wire, and

$$\max(\mathrm{depth}(w_t), \mathrm{depth}(w_b)) + 1$$

for the output wires of a balancer with input wires w_t and w_b. Define the depth of a balancer to be the depth of its output wires, and the depth of a network to be the maximum depth of any of its balancers. For example, the balancing network in Figure 3 has depth three. We group balancers of the same depth into *stages*. Define the *width* of a network to be the number of network input wires. For example, the balancing network in Figure 3 has width four.

We are only interested in the token counts on the output wires, not the routing of specific tokens. When we refer to inputs and outputs, we mean the token counts on input and output wires. For a balancing network of width n and depth d, let $\mathbf{x}^{(i)} = x_0^{(i)}, x_1^{(i)}, \ldots, x_{n-1}^{(i)}$ denote the token counts at the i-th stage, for $i = 0, \ldots, d$. If the network reaches a point where all tokens have been forwarded through to the output wires, it is said to be *quiescent*. In a quiescent network, the sum of the token counts for all the stages are equal, i.e.,

$$\sum_{j=0}^{n-1} x_j^{(0)} = \sum_{j=0}^{n-1} x_j^{(1)} = \cdots = \sum_{j=0}^{n-1} x_j^{(d)}.$$

We refer to the token counts at a given stage in a quiescent network as a *sequence*. Figure 4 shows the sequences for the example of Figure 3.

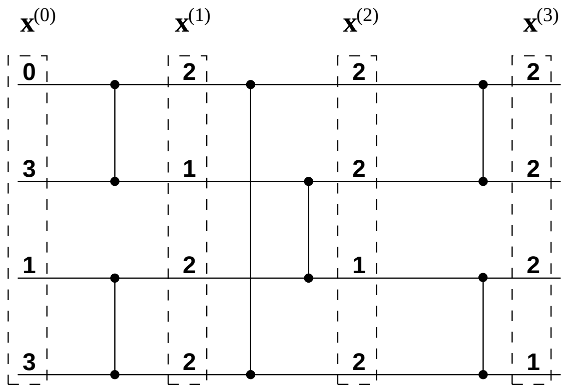

Figure 4. Sequences at every stage for the network of Figure 3.

We define the *distance* between two sequences $\mathbf{y} = y_0, y_1, \ldots, y_{n-1}$ and $\mathbf{y}' = y'_0, y'_1, \ldots, y'_{n-1}$ as one half the sum of the absolute value of the difference of their entries:

$$D(\mathbf{y}, \mathbf{y}') = \frac{1}{2} \sum_{i=0}^{n-1} |y_i - y'_i|.$$

We define two types of output sequences: smooth sequences and step sequences (see [4]).

- A sequence $y_0, y_1, \ldots y_{n-1}$ is *smooth* if and only if $|y_i - y_j| \leq 1$ for any i, j.

- A sequence $y_0, y_1, \ldots y_{n-1}$ is *step* if and only if $0 \leq y_i - y_j \leq 1$ for any $i < j$.

A *counting network* is a balancing network that produces step output sequences. A shared counter is implemented by assigning counter values based upon the output wire to which tokens are routed. Tokens arriving at output wire i of a counting network of width n are assigned values i, $i+n$, $i+2n$, ...

3. Fault Models

Several authors have investigated failure models for shared-memory systems, and proposed fault-tolerant constructions for shared objects [1][2][11][16]. In our model, the following errors can occur dynamically in the counting network data structure:

1. **Bit-Flip Fault**: A balancer's state is spuriously altered from `up` to `down`, or vice-versa.

2. **Bit-Access Fault**: A balancer's state, whether `up` or `down`, can no longer be accessed.

For shared-memory implementations, a bit-flip fault corresponds to an atomic write operation (by some outside agent) to a memory location holding a balancer's state variable: a one, representing `up`, is overwritten with a zero, representing `down`, or vice-versa. With a bit-access fault, the memory location holding a balancer's state variable behaves correctly until it suffers an atomic failure. Thereafter, it is inaccessible. Note that we do not consider process failures, which result in lost tokens. Also, we do not consider errors affecting the network wiring information (the topology of the network is static).

3.1. Bit-Flip Faults

A bit-flip fault can result in an imbalance in the token counts on a balancer's output wires. Figure 5 gives an example. In Part (a), the balancer forwards tokens 1, 2 and 3. At this point, the balancer's state is `down`, indicating that the next token should be forwarded to the bottom wire. However, a bit-flip fault occurs setting the balancer's state to `up`. This leads to an imbalance in the token counts, shown in Part (b).

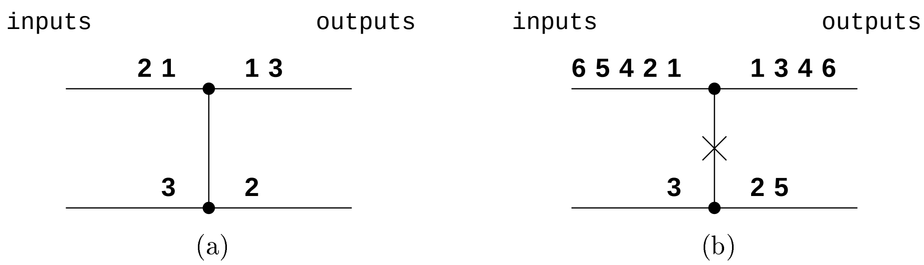

Figure 5. A balancer experiencing a bit-flip fault.

Denote by x_t and x_b the number of tokens received on a balancer's top and bottom input wires, respectively. Similarly, denote by y_t and y_b the number of tokens forwarded to its top and bottom output wires, respectively. A bit-flip fault alters the outputs of the balancer as follows:

$$y_t = \left\lceil \frac{x_t + x_b}{2} \right\rceil + f,\ y_b = \left\lfloor \frac{x_t + x_b}{2} \right\rfloor - f$$

for some $f \in \{-1, 0, 1\}$.

Consider a balancer with outputs on wires i and j of some stage of a balancing network. Suppose that the balancer does not experience any faults. Let the output sequence of the stage be $\mathbf{y} = y_0, y_1, \ldots, y_i, \ldots, y_j, \ldots, y_{n-1}$. If instead the balancer experiences a bit-flip fault, the output sequence of the stage is $\mathbf{y}' = y_0, y_1, \ldots, y_i', \ldots, y_j', \ldots, y_{n-1}$, where $y_i' = y_i + f$ and $y_j' = y_j - f$ for some $f \in \{-1, 0, 1\}$. Clearly, the distance between $\mathbf{y}$ and $\mathbf{y}'$ is less than or equal to 1. With k bit-flip faults in the stage, the distance between the two sequences is less than or equal to k.

3.2. Bit-Access Faults

Our strategy in coping with bit-access faults is to bypass inaccessible balancers. Thus, tokens are forwarded out along the same wire that they are received on (out the top if they are received on the top, or out the bottom if they are received on the bottom). Denote by x_t and x_b the number of tokens received prior to a bit-access fault on a balancer's top and bottom input wires, respectively; denote by x_t' and x_b' the number of tokens received after the fault on the balancer's top and bottom input wires, respectively. Denote by y_t and y_b the total number of tokens forwarded to its top and bottom output wires, respectively. We have

$$y_t = \left\lceil \frac{x_t + x_b}{2} \right\rceil + x_t', \quad y_b = \left\lfloor \frac{x_t + x_b}{2} \right\rfloor + x_b'.$$

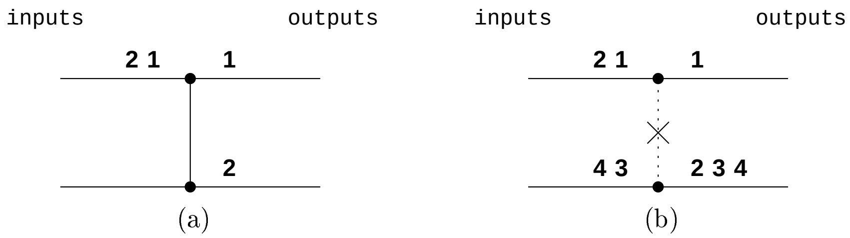

Figure 6. A balancer experiencing a bit-access fault.

With a bit-access fault, there may be an arbitrarily large imbalance in the outputs of a faulty balancer, even if the inputs to the balancer are balanced. Consider the example shown in Figure 6. In Part (a), the balancer forwards tokens 1 and 2. In

Part (b), the balancer's state can no longer be accessed due to a bit-access fault. Tokens 3 and 4 bypass the balancer, exiting on the bottom wire. Note that the inputs are balanced but the outputs are not.

3.3. Mapping Bit-Access Faults to Bit-Flip Faults

With some implementations, if the underlying fault behavior is that of bit-access faults, it may be possible to obtain the behavior of bit-flip faults by mapping inaccessible balancers to spare balancers. A spare balancer is given a random initial state. If this state is different from the original balancer's state, the situation is equivalent to a bit-flip fault. For shared-memory implementations, the mapping is accomplished by redirecting the pointers of the preceding balancers from the inaccessible balancer to the spare balancer.

4. Tolerating Faults

We propose two approaches for tolerating faults.

<u>Bit-Access Faults:</u>

The first approach is applicable for implementations experiencing bit-access faults. In Section 4.1, we describe a construction for a fault-tolerant balancer with $2(k+1)$ bits, capable of tolerating k bit-access faults. With this construction, we substitute a fault-tolerant balancer for every balancer in a counting network. Thus, we transform a counting network with depth $O(\log^2 n)$ into a k-fault-tolerant counting network with depth $O(k \log^2 n)$. We note that similar results could be obtained based on the constructions for fault-tolerant shared object presented by Afek et al. [2] and Jayanti et al. [11].

<u>Bit-Access and Bit-Flip Faults:</u>

The second approach is to append a *correction network* to a counting network, as shown in Figure 7. The counting network consists of balancers that may experience bit-flip faults, while the correction network consists of fault-tolerant balancers that may experience bit-access faults. In Section 4.2, we present an upper bound on the error in the token distribution resulting from bit-flip faults. In Section 4.4, we prove the following property for the correction network: given the output token distribution of a faulty counting network with a bounded error, the correction network produces a token distribution that is smooth. In order to tolerate k faults, the correction network consists of $k(\log n+1)$ stages of fault-tolerant balancers, each with $2(k+1)$ bits. Thus, a k-fault-tolerant construction consists of a counting network with depth $O(\log^2 n)$ and a correction network with depth $2k(k+1)(\log n+1)$.

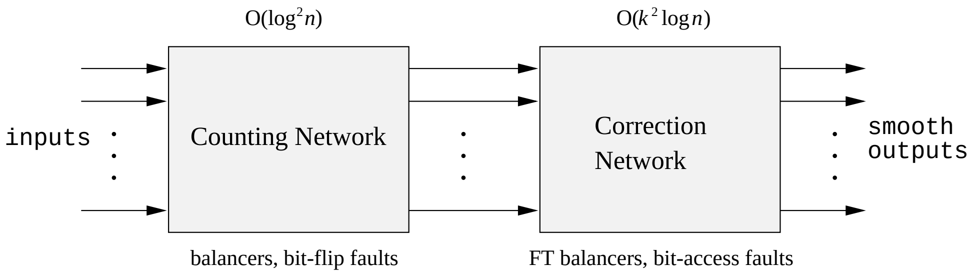

Figure 7. A correction network appended to a counting network.

4.1. Fault-Tolerant Balancer

In our construction, shown in Figure 8, a k-fault-tolerant balancer consists of $k+1$ *pseudo-balancers*, each with two bits of memory. The first bit describes its *state*: either `up` or `down`, indicating that the next token should be forwarded to the top or bottom output wire, respectively. The second bit describes its *status*: either it is a `leader` or a `follower`. Initially, the first pseudo-balancer is a leader while the others are followers. We assume that the pseudo-balancers experience bit-access faults. An inaccessible pseudo-balancer is bypassed, i.e., tokens are forwarded directly to the next pseudo-balancer along the same wire that they are received on. Tokens are colored with one of two colors: red indicating that they have been balanced, or green indicating that they have not. Tokens entering a fault-tolerant balancer are initially colored green.

Leader:

A leader balances tokens in the usual fashion. It accepts tokens on either of its input wires, and forwards them alternately to its top and bottom output wires, toggling its state from `up` to `down` or vice-versa. It colors all outgoing tokens red.

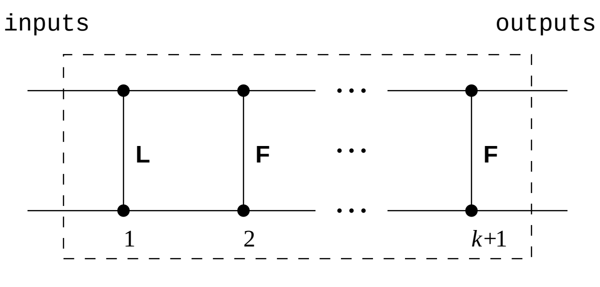

Figure 8. A fault-tolerant balancer (`L` = leader, `F` = follower).

Follower:

A follower's behavior differs for red and green tokens. A follower only accepts red tokens in order: first one from its top input wire, then one from its bottom input wire, and so on. As it receives red tokens, it toggles its state from `up` to `down`, or vice-versa, and forwards the tokens along the same wire that it receives them on. A green token is an indication that all pseudo-balancers before it have failed. Thus, as soon as a follower receives a green token on either input wire, it becomes a leader and starts routing tokens as described above.

The algorithm for pseudo-balancers is described in greater detail in the Appendix. An example is shown in Figure 9. Initially, the first pseudo-balancer is a leader and the second a follower. In Part (a), the first pseudo-balancer balances tokens 1 and 2, coloring them red. The second pseudo-balancer receives tokens 1 and 2 in order, and forwards them through. In Part (b), the first pseudo-balancer is no longer accessible due to a bit-access fault. Thus, tokens 3 and 4 bypass it along the bottom wire. When the second pseudo-balancer receives token 3, colored green, it becomes a leader and begins balancing the tokens that it receives.

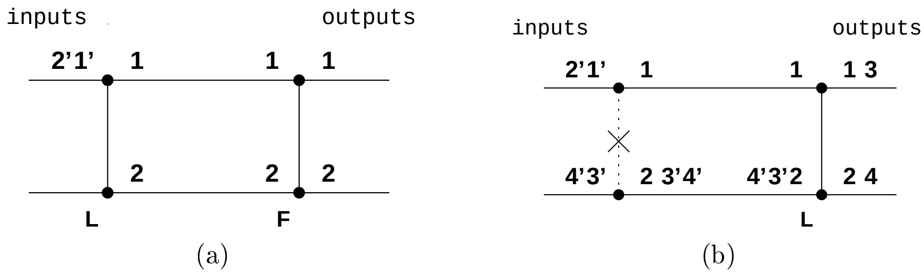

Figure 9. A fault-tolerant balancer experiencing a bit-access fault (`L` = leader, `F` = follower, n' = green token, n = red token).

THEOREM 1 *With at most k faults, the outputs of a k-fault-tolerant balancer are balanced.*

Proof: With at most k faults, at least one of the $k+1$ pseudo-balancers never fails. The outputs of this pseudo-balancer are balanced; all tokens that it forwards are colored red. Subsequent pseudo-balancers never become leaders, since they never receive green tokens. Followers forward tokens along the same wire that they received them on. Similarly, tokens remain on the same wire when bypassing inaccessible balancers. Thus, all tokens forwarded by a pseudo-balancer that never fails remain on the same wire. Therefore, the outputs of the fault-tolerant balancer are balanced. ∎

Note that there is fine-grained synchronization among processes shepherding tokens concurrently through a fault-tolerant balancer. If a follower receives a red token on the wrong wire (on the bottom wire if its state is `up`, or on the top wire if its state is `down`) then it will block the token. However, if this occurs, then the arrival of a token on the other wire is pending.

4.2. Error Bound for Bit-Flip Faults

We will show the following result: k bit-flip faults cause an error of at most k in the output token distribution of a balancing network. This bound applies irrespective of the number of tokens launched into the network. Figure 10 illustrates this result with Aspnes et al.'s `BITONIC` counting network [4]. A bit-flip fault occurs in the third balancer from the left. As a result, the output sequence becomes $3, 2, 1, 1$ instead of $2, 2, 2, 1$.

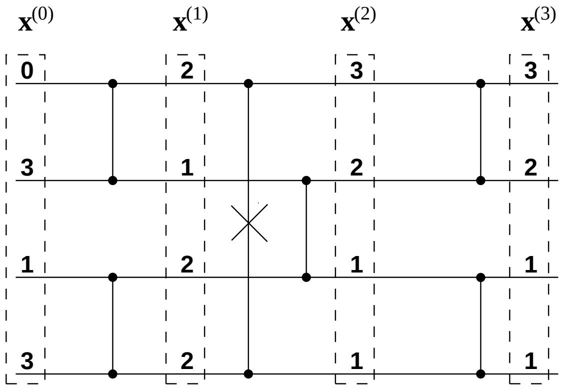

Figure 10. A `BITONIC[4]` network with a balancer experiencing a bit-flip fault.

The following analysis is a generalization of the error bound for sorting networks with faulty comparators by Yao & Yao [21]. Related work can also be found in the paper by Schimmler and Starke [17].

LEMMA 1 *Balancing the same entries in two sequences cannot increase the distance between them.*

Proof: Without loss of generality, consider sequences of length two: $\mathbf{x} = x_0, x_1$ and $\mathbf{y} = y_0, y_1$. Balancing x_0 and x_1 as well as y_0 and y_1 yields $\mathbf{x}' = x'_0, x'_1$ and $\mathbf{y}' = y'_0, y'_1$ where

$$x'_0 = \left\lceil \frac{x_0 + x_1}{2} \right\rceil, x'_1 = \left\lfloor \frac{x_0 + x_1}{2} \right\rfloor$$

and

$$y'_0 = \left\lceil \frac{y_0+y_1}{2} \right\rceil, y'_1 = \left\lfloor \frac{y_0+y_1}{2} \right\rfloor.$$

We consider four cases. In each case, we show that the distance between $\mathbf{x}'$ and $\mathbf{y}'$ is less than or equal to the distance between $\mathbf{x}$ and $\mathbf{y}$.

1. $x_0 + x_1$ is odd, $y_0 + y_1$ is odd:

$$\begin{aligned}
D(\mathbf{x}', \mathbf{y}') &= \frac{1}{2}\left|\left\lfloor \frac{x_0+x_1}{2} \right\rfloor - \left\lfloor \frac{y_0+y_1}{2} \right\rfloor\right| + \frac{1}{2}\left|\left\lceil \frac{x_0+x_1}{2} \right\rceil - \left\lceil \frac{y_0+y_1}{2} \right\rceil\right| \\
&= \frac{1}{2}\left|\frac{x_0+x_1-1}{2} - \frac{y_0+y_1-1}{2}\right| + \frac{1}{2}\left|\frac{x_0+x_1+1}{2} - \frac{y_0+y_1+1}{2}\right| \\
&= \frac{1}{2}|x_0+x_1-y_0-y_1| \\
&\le \frac{1}{2}|x_0-y_0| + \frac{1}{2}|x_1-y_1| \\
&= D(\mathbf{x}, \mathbf{y})
\end{aligned}$$

2. $x_0 + x_1$ is even, $y_0 + y_1$ is even:

   Omitted (similar to the previous case).

3. $x_0 + x_1$ is odd, $y_0 + y_1$ is even:

$$\begin{aligned}
D(\mathbf{x}', \mathbf{y}') &= \frac{1}{2}\left|\left\lfloor \frac{x_0+x_1}{2} \right\rfloor - \left\lfloor \frac{y_0+y_1}{2} \right\rfloor\right| + \frac{1}{2}\left|\left\lceil \frac{x_0+x_1}{2} \right\rceil - \left\lceil \frac{y_0+y_1}{2} \right\rceil\right| \\
&= \frac{1}{2}\left|\frac{x_0+x_1-1}{2} - \frac{y_0+y_1}{2}\right| + \frac{1}{2}\left|\frac{x_0+x_1+1}{2} - \frac{y_0+y_1}{2}\right| \\
&= \frac{1}{4}|x_0+x_1-y_0-y_1-1| + \frac{1}{4}|x_0+x_1-y_0-y_1+1| \\
&= \frac{1}{2}|x_0+x_1-y_0-y_1| \qquad \text{(since } x_0+x_1 \neq y_0+y_1\text{)} \\
&\le \frac{1}{2}|x_0-y_0| + \frac{1}{2}|x_1-y_1| \\
&= D(\mathbf{x}, \mathbf{y})
\end{aligned}$$

4. $x_0 + x_1$ is even, $y_0 + y_1$ is odd:

   Omitted (similar to previous case).

■

THEOREM 2 *Consider two identical balancing networks given the same input sequence. If the first is fault-free and the second experiences k bit-flip faults, then the distance between the output sequences of the two networks is less than or equal to k.*

Proof: Let the networks be of depth d. Let $\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(d)}$ and $\mathbf{y}^{(0)}, \ldots, \mathbf{y}^{(d)}$ be the sequences at successive stages in the fault-free and faulty networks, respectively.

With identical input sequences, we have $\mathbf{x}^{(0)} = \mathbf{y}^{(0)}$. A balancer experiencing a bit-flip fault in some stage i of the faulty network, where $1 \leq i \leq d$, increases the distance between $\mathbf{x}^{(i)}$ and $\mathbf{y}^{(i)}$ by at most one. By Lemma 1, fault-free balancers can only decrease the distance between $\mathbf{x}^{(i)}$ and $\mathbf{y}^{(i)}$. With k bit-flip faults, it follows that the distance between the output sequences of the two networks is at most k.

■

4.3. Smoothness Error

We define an error measure on a sequence with respect to the smoothness property.

Definition 1. The *smoothness error* of a sequence is the minimum distance between that sequence and a smooth sequence of the same length with the same total sum.

For example, the sequence $3, 2, 1, 1$ in Figure 10 has a smoothness error of one: if we subtract one token from the first entry and add this token to the third entry, we obtain the smooth sequence $2, 2, 2, 1$.

A fault-free counting network produces step output sequences. By Theorem 2, the distance between the output sequence of a counting network with k bit-flip faults and a step sequence with the same total sum is at most k. Since every step sequence is a smooth sequence, the smoothness error of the output sequence of a counting network with k bit-flip faults is at most k.

4.4. Correction Network

In this section, we describe the construction of a balancing network called a *correction* network with the following property: given an input sequence with a smoothness error of at most k, it produces a smooth output sequence. This network is appended to a counting network that may experience at most k bit-flip faults. Since the output sequence of the counting network has a smoothness error of less than or equal to k, the final output sequence from the correction network is smooth.

The correction network is constructed from blocks that we call `CORRECT[`n`]` networks. A `CORRECT[8]` network is shown in Figure 11. To tolerate k bit-flip faults in a counting network of width n, we append k copies of `CORRECT[`n`]`, as shown in Figure 12. Each copy has $\log n + 1$ stages. Note that the `CORRECT[`n`]` networks are built with k-fault-tolerant balancers.

Before describing the construction of the correction network, we prove the following claim.

CLAIM 1 *Balancing two entries of a sequence cannot increase the smoothness error.*

Proof: Let $\mathbf{x} = x_0, x_1, \ldots x_{n-1}$ be a sequence with smoothness error k. Suppose that we balance the i-th and j-th entries of $\mathbf{x}$, $0 \leq i < j < n$. This yields a sequence $\mathbf{x}' = x_0, x_1, \ldots, x'_i, \ldots, x'_j, \ldots, x_{n-1}$, where

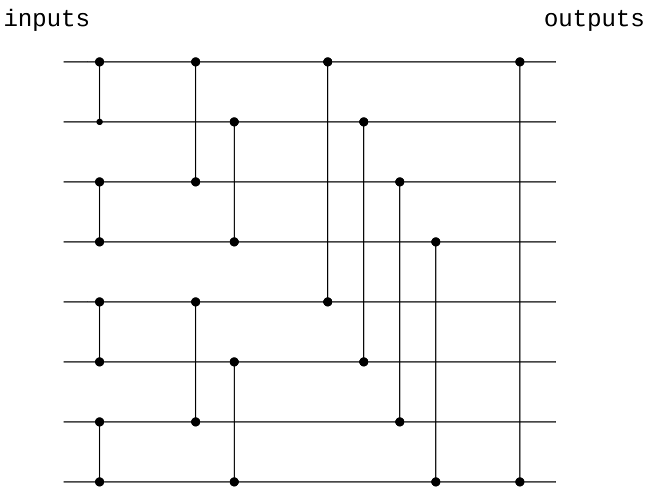

Figure 11. The `CORRECT[8]` network.

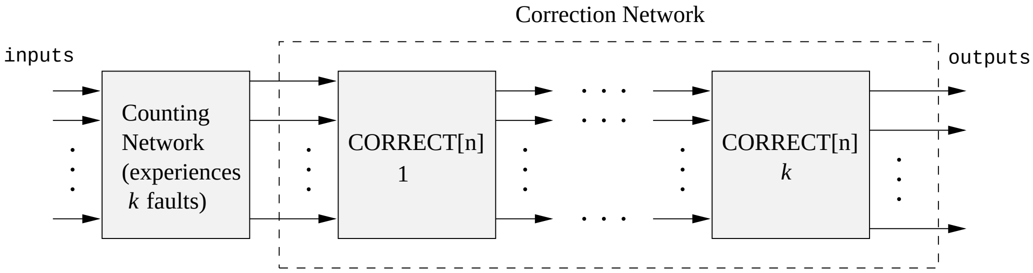

Figure 12. Correcting the output sequence of a counting network experiencing k bit-flip faults.

$$x'_i = \left\lceil \frac{x_i + x_j}{2} \right\rceil,\ x'_j = \left\lfloor \frac{x_i + x_j}{2} \right\rfloor.$$

Let the *target sequence* $\mathbf{s} = s_0, s_1, \ldots s_{n-1}$ be a smooth sequence with the same total sum as $\mathbf{x}$:

$$\sum_{i=0}^{n-1} x_i = \sum_{i=0}^{n-1} s_i.$$

Obviously, balancing any two entries of a smooth sequence yields a smooth sequence. So if we balance the i-th and j-th entries of $\mathbf{s}$, $0 \leq i < j < n$, the sequence $\mathbf{s}' = s_0, s_1, \ldots, s'_i, \ldots, s'_j, \ldots, s_{n-1}$ is smooth. According to Lemma 1, $D(\mathbf{x}', \mathbf{s}') \leq D(\mathbf{x}, \mathbf{s})$. Therefore, the smoothness error of $\mathbf{x}'$ is less than or equal to the smoothness error of $\mathbf{x}$. ■

4.4.1. `BUTTERFLY[`n`]` *network:* In order to construct a `CORRECT[`n`]` network, we require a building block called the `BUTTERFLY[`n`]` network. This network is constructed recursively as follows. For width two, it consists of a single balancer. For width n, where $n = 2^m$ for some $m > 1$, the network consists of two `BUTTERFLY[`$n/2$`]` networks with a balancer placed between output wire i of the top network and output wire i of the bottom network, for each $i = 0, \ldots n/2 - 1$ (see Figure 13).

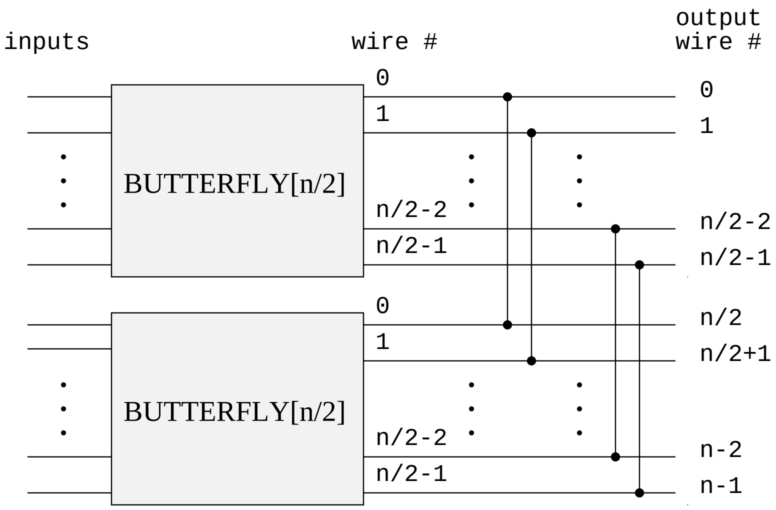

Figure 13. Recursive construction of the `BUTTERFLY[`n`]` network.

CLAIM 2 *The output value on wire 0 of a* `BUTTERFLY[`n`]` *network is the largest in the output sequence, and the output value on wire $n - 1$ is the smallest.*

Proof: The proof is by induction on the network width. A network of width two consists of a single balancer, so the claim is trivially true. Assume that the claim holds for a network of width $n/2$, where $n = 2^m$ for some $m > 1$. For a network of width n, every balancer in the final stage receives one input value from the top `BUTTERFLY[`$n/2$`]` network and the other from the bottom `BUTTERFLY[`$n/2$`]` network. In particular, wire 0 is the output of a balancer that receives inputs from output wires 0 of the top and bottom `BUTTERFLY[`$n/2$`]` networks. By the induction hypothesis, these wires have the largest output values from each of the two `BUTTERFLY[`$n/2$`]` networks. Thus, the output value on wire 0 is the largest in the output sequence. By a symmetrical argument, the output value on wire $n-1$ is the smallest. Therefore, the claim holds for a network of width n. By induction, the claim is true for all network widths N, where $N = 2^m$ for some $m \geq 1$. ■

Note that it is necessary to balance all the corresponding outputs from the two `BUTTERFLY[`$n/2$`]` networks, in addition to the outputs on wires 0 and $n-1$; otherwise an unbalanced output could be the largest or the smallest.

4.4.2. `CORRECT[`n`]` *network:* The `CORRECT[`n`]` network is obtained by placing a balancer between wires 0 and $n-1$ at the output of a `BUTTERFLY[`n`]` network, as shown in Figure 14.

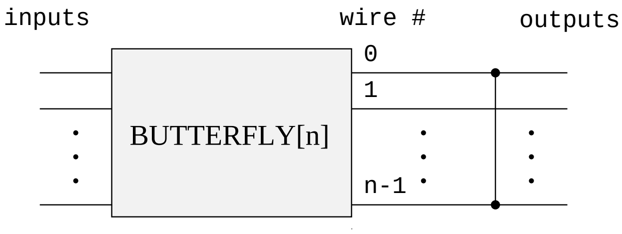

Figure 14. Construction of the `CORRECT[`n`]` network.

THEOREM 3 *Given an input sequence of length n with a smoothness error of at most k, for some $k \geq 0$, k copies of the* `CORRECT[`n`]` *network produce a smooth output sequence.*

Proof: Let $\mathbf{w} = w_0, \ldots w_{n-1}$ and $\mathbf{y} = y_0, \ldots y_{n-1}$ be the inputs and outputs of a `BUTTERFLY[`n`]` network, respectively. By Claim 1, the smoothness error of $\mathbf{y}$ is less than or equal to that $\mathbf{w}$. By Claim 2, y_0 and y_{n-1} are the largest and smallest entries of $\mathbf{y}$, respectively. In the final stage of a `CORRECT[`n`]` network, we balance y_0 and y_{n-1} which yields an output sequence $\mathbf{y}' = y'_0, y_1, \ldots, y'_{n-1}$, where

$$y'_0 = \left\lceil \frac{y_0 + y_{n-1}}{2} \right\rceil, \; y'_{n-1} = \left\lfloor \frac{y_0 + y_{n-1}}{2} \right\rfloor,$$

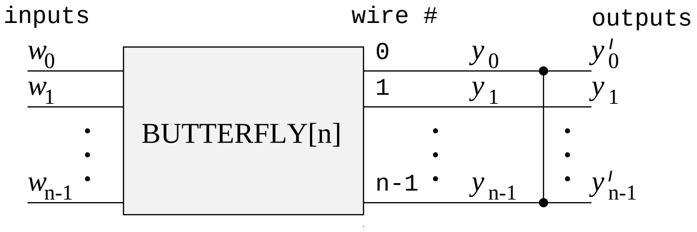

Figure 15. Construction of the `CORRECT[`n`]` network.

as shown in Figure 15.

Our strategy is to show that each copy of the `CORRECT[`n`]` network reduces the smoothness error by at least one, unless the smoothness error is zero.

Intuitive Argument:

Suppose that not all the entries of $\mathbf{y}$ are equal. Let m be the mean value of the entries of $\mathbf{y}$. Without loss of generality, suppose that $y_0 = m + h$ and $y_{n-1} = m - l$ for some $h \geq l > 0$. Define an error measure E as one half the sum of the absolute value of the difference between each entry and the mean:

$$E = \frac{1}{2} \sum_{i=0}^{n-1} |y_i - m|.$$

Thus, y_0 and y_{n-1} contribute $\frac{1}{2}(h+l)$ to the error. Since, $h \geq l$, y'_0 and y'_{n-1} are both greater than or equal to m. Their contribution to the error is

$$\begin{aligned} &\frac{1}{2}\left(\left\lceil \frac{y_0 + y_{n-1}}{2} \right\rceil - m + \left\lfloor \frac{y_0 + y_{n-1}}{2} \right\rfloor - m\right) \\ &= \frac{1}{2}\left(y_0 + y_{n-1} - 2m\right) \\ &= \frac{1}{2}\left(h - l\right). \end{aligned}$$

We see that the contribution of these entries to the error is reduced from $\frac{1}{2}(h+l)$ to $\frac{1}{2}(h-l)$. Thus, balancing the largest and smallest entries reduces the error by an amount $l > 0$.

Formal Argument:

The formal argument is essentially the same as the one above, but is complicated by the fact that smoothness is defined in terms of discrete values. Let the *target sequence* $\mathbf{s} = s_0, s_1, \ldots s_{n-1}$ be a smooth sequence with the same total sum as $\mathbf{y}$:

$$\sum_{i=0}^{n-1} y_i = \sum_{i=0}^{n-1} s_i.$$

We consider two cases.

1. The total sum of $\mathbf{y}$ is not evenly divisible by its length n. In this case, $\mathbf{s}$ has two distinct values c_L and c_H, where $c_H = c_L + 1$. For example, if $\mathbf{y}$ has length 8 with total sum 29, then $\mathbf{s}$ consists of 3's and 4's.

2. The total sum of $\mathbf{y}$ is evenly divisible by its length n. In this case, all the entries in $\mathbf{s}$ are equal. For example, if $\mathbf{y}$ has length 8 with total sum 32, then $\mathbf{s}$ consists of all 4's.

For both cases, we show that balancing y_0 and y_{n-1} reduces the smoothness error by at least one, unless the smoothness error is zero.

1. The total sum of $\mathbf{y}$ is not evenly divisible by its length n. In this case, $\mathbf{s}$ has two distinct values c_L and c_H, where $c_H = c_L + 1$.

   We define two error measures on the entries of $\mathbf{y}$:

   $$d_L = \sum_{y_i \le c_L} (c_L - y_i),$$

   $$d_H = \sum_{y_i \ge c_H} (y_i - c_H).$$

   Note that $\mathbf{y}$ is smooth if and only if $d_L = d_H = 0$. We show that balancing y_0 and y_{n-1} decreases d_L by at least one if $d_L > 0$ and decreases d_H by at least one if $d_H > 0$.

   Since y_0 and y_{n-1} are the largest and smallest entries of $\mathbf{y}$, respectively, it follows that y_0 must be greater than or equal to c_H, and y_{n-1} must be less than or equal to c_L. Let $y_0 = c_H + h$ and $y_{n-1} = c_L - l$, where $l, h \ge 0$. Note that y_0 contributes zero to d_L and contributes h to d_H, while y_{n-1} contributes zero to d_H and contributes l to d_L. Note also that $h > 0$ if and only if $d_H > 0$, and $l > 0$ if and only if $d_L > 0$.

   After balancing y_i and y_j, we obtain new entries y_i' and y_j':

   $$y_i' = \left\lceil \frac{y_i + y_j}{2} \right\rceil = \left\lceil \frac{c_H + h + c_L - l}{2} \right\rceil,$$
   $$y_j' = \left\lfloor \frac{y_i + y_j}{2} \right\rfloor = \left\lfloor \frac{c_H + h + c_L - l}{2} \right\rfloor.$$

   We now consider three possible subcases.

(A) $l < h$: y'_i and y'_j are both greater than or equal to c_H, so they contribute zero to d_L. Thus, d_L is decreased by l. Together y'_i and y'_j contribute $c_H + h + c_L - l - 2c_H = h - l - 1$ to d_H. Thus, d_H is decreased by $l + 1$.

(B) $l > h$: Omitted (similar to the previous case).

(C) $l = h$: $y'_i = c_H$ and $y'_j = c_L$, so they both contribute zero to d_L and d_H. Thus, d_H is decreased by h and d_L is decreased by l.

Thus, balancing y_0 and y_{n-1} reduces d_L by at least one if $d_L > 0$, and reduces d_H by at least one if $d_H > 0$. Therefore, it reduces the smoothness error of $\mathbf{y}$ by at least one, unless the smoothness error of $\mathbf{y}$ is zero.

2. The total sum of $\mathbf{y}$ is evenly divisible by its length n.

   Omitted (similar to Case 1).

Thus, each `CORRECT[`n`]` network reduces the smoothness error by at least one, unless the smoothness error is zero. Therefore, given an input sequence with a smoothness error of at most k, it follows that k copies of the `CORRECT[`n`]` network produce a smooth output sequence. ■

5. Discussion

The construction described in Section 4.4 ensures that the token counts on all the output wires differ by at most one. This smoothness property is weaker than the step property of counting networks in the sense that every step sequence is a smooth sequence, but not vice-versa. However, for applications such as load-balancing, a smoothing network is just as effective as a counting network.

Fault tolerance has important advantages for many applications. For instance, in space applications, faults in memory are common due to radiation effects. In distributed systems, nodes can become inaccessible due to halting failures or network congestion. We have presented an upper bound on the error resulting from faults in balancing networks. The bound states that each fault causes an error of at most one in the output token distribution. We have presented a practical method for tolerating k faults in a counting network, with an increase in the depth of $2k(k+1)(\log n + 1)$ for a network of width n. This is small compared to the depth of the counting network itself, which is $O(\log^2 n)$ for all practical constructions. Future work is needed to derive lower bounds on the depth of correction networks, and to extend these concepts to *diffracting trees*, a variation of counting networks recently proposed by Shavit et al. [18][19][20].

Appendix

A.1. Code for Implementing Pseudo-Balancers

Leader:

```
procedure leader:
do forever
    receive a token on the top or bottom input wire;
    if state = up then
        forward a red token to the top output wire;
        state := down;
    else
        forward a red token to the bottom output wire;
        state := up;
    end
end
```

Follower:

```
procedure follower:
do forever
    select
        either receive a green token on the top or bottom input wire;
        or if state = up then receive a red token on the top input wire;
        or if state = down then receive a red token on the bottom input wire;
    end
    if state = up then
        forward a red token to the top output wire;
        state := down;
        if the token received was green then
            begin executing the procedure leader;
        end
    else
        forward a red token to the bottom output wire;
        state := up;
        if the token received was green then
            begin executing the procedure leader;
        end
    end
end
```

A.2. Pseudo-Balancers in Shared-Memory Systems

In shared-memory architectures, a process shepherding a token through a fault-tolerant balancer keeps a knowledge of the token's color (green or red) as well as which wire it is on (top or bottom). As it travels through the fault-tolerant balancer, it acquires simultaneous locks on both the status and state bits for each pseudo-balancer.

1. If the status is set to `leader`, then it sets the token's wire according to the state, sets the token's color to red, toggles the state, releases both locks and proceeds to the next pseudo-balancer.

2. Else if the status is set to `follower` and the token is colored green, then it sets the status to `leader`, sets the wire according to the state, sets the token's color to red, toggles the state, releases both locks and proceeds to the next pseudo-balancer.

3. Else if the status is set to `follower`, the token is colored red, the state is set to `up` and the token is on the top wire, or the state is set to `down` and the token is on the bottom wire, then it toggles the state, releases both locks and proceeds to the next pseudo-balancer.

4. Else if the status is set to `follower`, the token is colored red, the state is set to `up` and the token is on the bottom wire, or the state is set to `down` and the token is on the top wire, then it releases both locks and blocks until one of the preceding conditions is met.

References

1. Y. Afek, M. Merritt and G. Taubenfeld, "Benign Failure Models for Shared-Memory", Lecture Notes in Computer Science, Vol. 725, pp. 69-83, 1993.
2. Y. Afek, D.S. Greenberg, M. Merritt and G. Taubenfeld, "Computing with Faulty Shared Objects", *Journal of the ACM*, Vol. 42., No. 6, pp. 1231-1274, 1995.
3. E. Aharonson and H. Attiya, "Counting Networks with Arbitrary Fan-Out", *Distributed Computing*, Vol. 8, No. 4, pp. 163-169, 1995.
4. J. Aspnes, M. Herlihy and N. Shavit, "Counting Networks", *Journal of the ACM*, Vol. 41, No. 5, pp. 1020-1048, 1994.
5. C. Busch and M. Mavronicolas, "A Combinatorial Treatment of Balancing Networks", *Journal of the ACM*, Vol. 43, No. 5, pp. 794-839, 1996.
6. C. Busch and M. Mavronicolas, "The Strength of Counting Networks", *Proceedings 15th ACM Symp. Principles of Distributed Computing* (Abstract), p. 300, 1996.
7. E.W. Felton, A. LaMarca and R. Ladner, "Building Counting Networks from Larger Balancers", Tech. Rep. 93-04-09, Univ. Washington.
8. N. Hardavellas, D. Karakos and M. Mavronicolas, "Notes on Sorting and Counting Networks", *Proceedings 7th International Workshop on Distributed Algorithms*, Lecture Notes in Computer Science, Vol. 725, pp. 234-248, 1993.
9. M. Herlihy, N. Shavit and B.-H. Lim, "Scalable Concurrent Counting", Vol. 13, No. 4, *ACM Trans. Computer Systems*, pp. 343-364, 1995.
10. M. Herlihy, N. Shavit and O. Waarts, "Linearizable Counting Networks", *Distributed Computing*, Vol. 9, No. 4, pp. 193-203, 1996.

11. P. Jayanti, T.D. Chandra and S. Toueg, "Fault-Tolerant Wait-Free Shared Objects", *Journal of the ACM*, Vol. 45, No. 3, pp. 451-500, 1998.
12. M. Klugerman and C. G. Plaxton, "Small-Depth Counting Networks", *Proceedings 24th ACM Symp. Theory of Computing*, pp. 417-428, 1992.
13. N. Lynch, N. Shavit, A. Shvartsman and D. Touitou, "Counting Networks are Practically Linearizable", *Proceedings 15th ACM Symp. Principles of Distributed Computing*, pp. 280-289, 1996.
14. M. Mavronicolas, "Balancing Networks - State-of-the-Art", *Information Sciences*, Vol. 97, No. 1-2, pp. 125-157, 1997.
15. M. Mavronicolas, M. Papatriantafilou and P. Tsigas,""The Impact of Timing on Linearizability in Counting Networks", *Proceedings 11th International Parallel Processing Symp.*, 1997.
16. A. Orda and M. Merritt, "Efficient Test-and-Set Constructions for Faulty Shared-Memory", *Information Processing Letters*, Vol. 62, No. 1, pp. 41-46, 1997.
17. M. Schimmler and C. Starke, "A Correction Network for N-Sorters", *SIAM J. Comput.*, Vol. 18, No. 6, pp. 1179-1187, 1989.
18. N. Shavit and A. Zemach, "Diffracting Trees", *ACM Trans. Computer Systems*, Vol. 14, No. 4, pp. 385-428, 1996.
19. N. Shavit, E. Upfal, and A. Zemach, "A Steady State Analysis of Diffracting Trees", *Proceedings 8th ACM Symp. Parallel Algorithms and Architectures*, pp. 33-41, 1996.
20. N. Shavit and D. Touitou, "Elimination Trees and the Construction of Pools and Stacks", *Theory of Computing Systems*, Vol. 30, No. 6, pp. 645-670, 1997.
21. A. Yao and F. Yao, "On Fault-Tolerant Networks for Sorting", *SIAM J. Computing*, Vol. 14, No. 1, pp. 120-128, 1985.