Counting Networks are Practically Linearizable*

Nancy Lynch † Nir Shavit ‡ Alex Shvartsman§ Dan Touitou ¶

Abstract

Counting networks are a class of concurrent structures that allow the design of highly scalable concurrent data structures in a way that eliminates sequential bottlenecks and contention. Linearizable counting networks assure that the order of the values returned by the network reflects the real-time order in which they were requested. We argue that in many concurrent systems the worst case scenarios that violate linearizability require a form of timing anomaly that is uncommon in practice. The linear time cost of designing networks that achieve linearizability under all circumstances may thus prove an unnecessary burden on applications that are willing to trade-off occasional non-linearizability for speed and parallelism.

This paper presents a very simple measure that is *local* to the individual links and nodes of the network, and that quantifies the extent to which a network can suffer from timing anomalies and still remain linearizable. Perhaps counter-intuitively, this measure is independent of network depth. We use our measure to mathematically support our experimental results: that in a variety of normal situations tested on a simulated shared memory multiprocessor, the *bitonic* counting networks of Aspnes, Herlihy, and Shavit are "for all practical purposes" linearizable.

*This work was supported by the following grants: ARPA N00014-92-J-4033 and F19628-95-C-0118, NSF 922124-CCR, and ONR-AFOSR F49620-94-1-01997.

†Massachusetts Institute of Technology, Laboratory for Computer Science. Email: `lynch@theory.lcs.mit.edu`.

‡Massachusetts Institute of Technology and Tel-Aviv University. Contact author: MIT Laboratory for Computer Science, 545 Technology Square, NE43-340, Cambridge, MA 02139. Email: `shanir@theory.lcs.mit.edu`. The work of this author was also supported by grant CCR 9520298.

§Massachusetts Institute of Technology, Laboratory for Computer Science. Email: `alex@theory.lcs.mit.edu`.

¶Tel-Aviv University, Department of Computer Science. Email: `danidin@cs.tau.ac.il`.

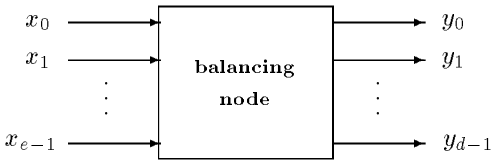
1 Introduction

Counting networks [4] are a class of concurrent structures that allow the design of highly scalable concurrent data structures in a way that eliminates sequential bottlenecks and contention. A recently developed form of counting network called a Diffracting Tree [21] has been shown to scale especially well, and has low latency since its depth is less than logarithmic in the number of processors n.

The notion of *linearizability* due to Herlihy and Wing [13] is the requirement that the values chosen by a concurrent object reflect the real-time order in which they were requested. The use of linearizable data abstractions greatly simplifies both the specification and the proofs of multiple instruction/multiple data (MIMD) shared memory algorithms. As explained in [13], linearizability generalizes and unifies a number of *ad-hoc* correctness conditions in the literature, and it is related to (but not identical with) correctness criteria such as sequential consistency [16] and strict serializability [19].

Herlihy, Shavit, and Waarts defined the class of linearizable counting networks [12], networks that assure that the order of the values returned by the network reflects the real-time order in which they were requested. Linearizable counting lies at the heart of concurrent timestamp generation, as well as concurrent implementations of shared counters, FIFO buffers, priority queues and similar data structures. Unfortunately, for both the *bitonic* networks of Aspnes, Herlihy, and Shavit [4] and the Diffracting Trees of Shavit and Zemach [21], there exist worst case asynchronous schedules in which linearizability is violated. In [12] linear depth linearizable counting network constructions were presented, and it was proven that this depth is inescapable: any low contention counting network that is linearizable in all executions must have linear depth, and hence is bound to have rather poor latency.

We argue that in many concurrent systems (e.g., distributed OS kernels or systems with limited multithreading) the worst case scenarios that violate linearizability require a form of timing anomaly that is uncommon in practice. The linear time cost of designing networks achieving linearizability under all circumstances may thus prove an unnecessary burden on applications that are willing to trade-off occasional non-



$\sum_{i=0}^{e-1} x_i \geq \sum_{i=0}^{d-1} y_i$

$0 \leq y_i - y_j \leq 1$ for any $i < j$

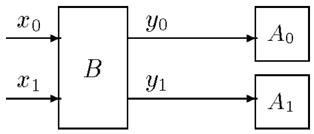
Figure 1: Balancing node and its input-output properties.

linearizability for speed and parallelism. Yet it is important to clearly characterize the parameters governing linearizability so that an intelligent trade-off decision can be made.

Our Contributions: We present a very simple measure that is *local* to the individual links and nodes of the network, and that quantifies the extent to which a network can suffer from timing anomalies and still remain linearizable. The measure does not depend on network size. We use our measure to mathematically support our experimental results: that in a variety of normal situations tested on a simulated shared memory multiprocessor, the bitonic counting networks of Aspnes, Herlihy, and Shavit are "for all practical purposes" linearizable. By this we mean that over a wide range of concurrency levels tested, even if one skews system timings by introducing large timing variations among processes, the network rarely exhibits violations of linearizability.

These results are especially interesting, for even a simple counting network of depth one easily exhibits nonlinearizable behavior.

Example: Consider the scenario for a counting network consisting of the balancer B and two atomic counters A_0 and A_1 with initial values 0 and 1, and that count by 2.



Token T_0 enters the balancer via x_0, exits via y_0, and then is delayed. Token T_1 enters via x_0 and exits via y_1 and obtains the value 1 from the counter A_1. Token T_2 enters via x_0 and exits via y_0 and obtains the value 0 from the counter A_0. Finally T_0 obtains the value 2 from A_0.

Here the behavior is not linearizable because the traversal of the network by T_1 completely precedes T_2, yet T_2 returns a lower counter value. □

A common structuring property of almost all published counting networks [1, 4, 3, 12, 14, 15, 10, 7, 20, 21] is *uniformity*: each node of the network lies on some path from inputs to outputs and all paths from inputs to outputs have equal lengths.

Our measure is as follows. Let c_1 be the minimum time that it takes for a token to traverse a wire from balancer to balancer, let c_2 be the maximum such time, and assume that balancer transitions are instantaneous. This timing model is general enough to capture both message passing and shared memory implementations using the same frameworks as [4, 21].

We prove (Section 3) the following for any uniform counting network, whether explicitly constructible or not:

- If $c_2 \leq 2 \cdot c_1$ then the network is linearizable. This is so regardless of the network depth.
- If $c_2 = k \cdot c_1$, where $k > 2$ then the network is linearizable if for any two tokens traversing the network their traversals either overlap or they are separated by time $t > h \cdot k(c_2 - 2 \cdot c_1)$, where h is the depth of the network.
- If a constant $k > 2$ is known *a priori*, such that $c_2 = k \cdot c_1$, then given a counting network of depth h we can extend this network by prefixing each of its inputs with $h(k-2)$ 1-input 1-output nodes so that the resulting network is a linearizable network of depth $O(h)$.

We also show (Section 4) that counting (diffracting) trees and bitonic counting networks are not linearizable for $c_2 > 2 \cdot c_1$, and we exhibit executions with large numbers of non-linearizable operations.

Finally, we provide experimental results collected for an Alewife [2] shared-memory multiprocessor simulated using Proteus [6], and use the c_2/c_1 ratio to explain bitonic counting network and diffracting tree linearizability properties (Section 5).

The full paper will be available on WWW in `http://theory.lcs.mit.edu/tds/papers.html`.

2 Models and definitions

Consider balancing networks consisting of acyclically wired multiple-input-multiple-output balancers or *balancing nodes* in the style of Aharonson and Attiya [1] and Felten, LaMarca, and Ladner [10] (Figure 1). We let x_i (respectively y_i) stand for the name of the input

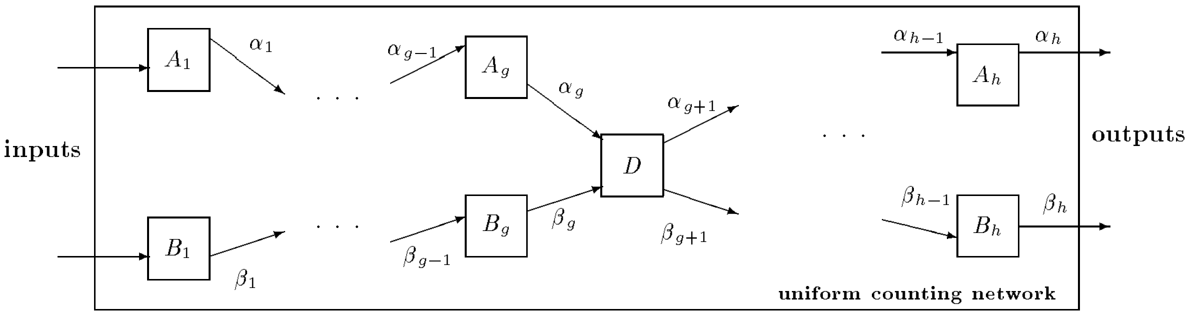


Figure 2: Equal length paths lead to any node in a uniform counting network.

port (output port) and its value for the number of tokens that have entered (exited) via that port. Each balancing node has a *step property* on its ordered outputs: in any state of the node, its outputs $y_0, y_1, \cdots, y_{d-1}$ satisfy $0 \leq y_i - y_j \leq 1$ for any $i < j$, where d is the number of outputs. Tokens are routed through a balancing node in a way that preserves the step property on its outputs. A balancing node does not generate any tokens spontaneously, but only routes tokens from the inputs to the outputs: $\sum_{i=0}^{e-1} x_i \geq \sum_{i=0}^{d-1} y_i$, where the equality is achieved in a state, called *quiescent*, in which all tokens have been routed and no new tokens arrive. This model is consistent with both the message passing and shared memory toggle bit based balancer implementations [4], and diffracting balancers [21], as all can be expressed in terms of balancers with atomic transitions.

Balancing or counting operations are processed in the form of tokens that are routed through a network. A *quiescent* state of a balancing network with v input ports $X_0, X_1, \ldots, X_{v-1}$ and w output ports $Y_0, Y_1, \ldots, Y_{w-1}$ is defined as the state when all tokens that have ever entered it have already exited. A *counting network* with w outputs is a balancing network that satisfies the following *step property*:

In any quiescent state, $0 \leq Y_i - Y_j \leq 1$ *for any* $i < j$.

The step property of counting networks is the cornerstone of the claims and proofs we will present.

The actual state transition of a balancer, i.e., the passing of a token from the node's input port to its output port, will be modeled as an instantaneous event. While balancer transitions are instantaneous, transitions along a link connecting an output port of one node to an input port of another are not. However, we assume that there is some c_1 that is the *lower bound* on time it takes for a token to traverse a link between balancers. Similarly there exists c_2 that is the *upper bound* on such time. Note that such links are also used to connect output ports of balancing nodes to the output ports of the network they are a part of. On the other hand, the input ports of a network are identified with the input ports of the nodes attached to the network inputs. Such nodes are called *input nodes*.

The output ports (or outputs) of a network are connected to atomic counters. Since these counters have a single input, we identify the counter attached to the output port Y_i by the name of the port. The tokens exiting from port Y_i are consecutively assigned the numbers $i, i+w, i+2w$, etc. The number assigned to a token by a counter is called the token's *returned value*.

Definition 2.1 A counting network is *uniform*, if each node of the network lies on some path from inputs to outputs, and all paths from inputs to outputs have equal lengths.

We define the *depth* of a uniform counting network as the number of links between any input node and output counter. The time t it takes for a token to traverse a uniform network of depth h is bounded by: $h \cdot c_1 \leq t \leq h \cdot c_2$. It is easy to see, from the above definition, that for each node D, the lengths all paths from the input nodes to D are equal and the lengths of all paths from D to the output nodes are equal (see Figure 2 – there and in the remaining figures, we do not show the counters attached to the outputs). For $1 \leq i \leq h$ we also define *layer* i of a network to be the collection of nodes whose distance from the inputs is $i-1$.

In the proofs, and without the loss of generality, we sequentially number the tokens traversing the network according to the time of their entry (ties are broken arbitrarily).

An execution of a network is a sequence $E = e_1, e_2, \ldots$ of instantaneous transition events $e_i = \langle T, D \rangle$ with the meaning of "token T traverses node D". Here D ranges over the balancing nodes and the atomic counters.

We associate *history variables* with tokens and nodes to capture their *implicit knowledge* about the execution. The history variables are sets of token ids. A history variable H_T is associated with each token T, and H_D with each node D. For every execution E the values of these variables are computed inductively as follows, where H_D^i and H_T^i denote the values of H_D and H_T after the event e_i:

- At the beginning of the execution, we define $H_D^0 = \emptyset$ and $H_T^0 = \{T\}$. In other words, at the beginning of the execution the knowledge of every node is an empty set and the knowledge of every token is the token's own id.
- The inductive step is as follows: Let $e_i = \langle T, D\rangle$, then $H_D^i = H_T^i = H_T^{i-1} \cup H_D^{i-1}$. Intuitively, the token T and the node D combine their knowledge as the result of e_i.

  For every other token $T' \neq T$ and node $D' \neq D$, $H_{T'}^i = H_{T'}^{i-1}$ and $H_{D'}^i = H_{D'}^{i-1}$.

Definition 2.2 A *timing schedule* S for an execution of a uniform network of depth h and input width v is a triplet $\langle K, L, Q\rangle$. K is the set of token ids produced by numbering the tokens based on their arrival times. $L : K \rightarrow \{x_i : 0 \leq i < v\}$ is the function such that for a token T_k, $L(k)$ is the input node on which the token enters the network. $Q : K \times [1..(h+1)] \rightarrow \mathcal{N}$ is the function such that $Q(k, j)$ is the real time instant when the token k passes through a node in the layer j of the network.

Adapting Herlihy and Wing definition [13] to counting networks:

Definition 2.3 A counting network is *linearizable* if for any execution, when two tokens traverse the network one after another without overlap, the earlier token obtains a smaller value than the later one.

Definition 2.4 Given an execution of a counting network, we say that some operation O is *non-linearizable*, if there exists some other operation O' completely preceding O in time that returns a higher counter value than O. The *fraction* of non-linearizable operations in an execution is the number of non-linearizable operations divided by the total number of operations of an execution.

3 A linearizability characterization for counting networks

In this section and the next, we now show that the ratio c_2/c_1 plays a key role in determining whether a uniform counting network is linearizable. In this section we show a very general condition implying that a uniform network is linearizable. In the following section we construct execution scenarios that prove that our analysis is tight for at least counting (diffracting) trees and bitonic networks.

We prove several lemmas that lead to the main result that uniform networks are linearizable for $c_2 \leq 2c_1$. The first lemma shows that in every counting network, when a token returns a certain value having traversed the network, it has implicit knowledge about the "existence" of a certain minimum number of other tokens.

Lemma 3.1 Let N be a counting network with w output ports $Y_0, \ldots, Y_{w-1}$. If the token T is the a^{th} token to exit on Y_i, then $|H_T| \geq w(a-1)+i+1$.

Proof: We sketch a proof by contradiction. We start by defining the notion of events *influencing* other events. For a pair of events e and e' in an execution E, we say that e *influences* e' if there is sequence of events $S = e_1, e_2, \ldots e_n$ such that (1) S is a subsequence of E, (2) $e = e_1$ and $e_n = e'$ and (3) for every $k = 1 \ldots n-1$ if $e_k = \langle T_k, D_k\rangle$ and $e_{k+1} = \langle T_{k+1}, D_{k+1}\rangle$, then either $T_k = T_{k+1}$ or $D_k = D_{k+1}$.

We now assume that there exists an execution E, in which T is the a^{th} token to exit on Y_i, but $|H_T| < w(a-1)+i+1$. We fix E and construct a new execution E' in the following way: Let E' be the subsequence of E consisting of all events involving T, and all the events that influence these events. From the definition of implicit knowledge, it is clear that E' contains events involving only the tokens found in H_T during the execution. We can show (the details are in the full paper) that E' is a possible execution of the counting network. In E', T is still the a^{th} token to exit on Y_i. Since only the tokens of H_T participate in E', every completion of E' in which no new token enters the network, will lead to a quiescent state with the step property violated. □

The next lemma shows that the implicit knowledge in the history variables can only reflect information propagation at the maximum rate of c_1 time units per link.

Lemma 3.2 Let N be a uniform counting network of depth h. For any execution $E = e_1, e_2, \ldots$, if $e_k = \langle T, D\rangle$ occurs at the time t, then H_D^k contains only tokens that have entered the network by the time $t - g\cdot c_1$, where g is the distance from D to an input node.

Proof: By induction on g: The base case for $g = 0$ is trivial. Assume lemma holds for $g-1$. We now show it holds for g.

Let $e_k = \langle T, D\rangle$ be the event of token T traversing node D at time t having traversed g links for some initial execution sequence $E = e_1, e_2, \ldots e_k$. From the definition of historical knowledge, $H_T^k = H_T^{k-1} \cup H_D^{k-1}$.

Consider the tokens in H_T^{k-1}. This set reflects T's knowledge after traversing $g-1$ links. By the induction hypothesis and because it takes at least c_1 time to traverse a link, all tokens in H_T^{k-1} have entered the network not later than time $(t-c_1)-(g-1)c_1 = t-g\cdot c_1$.

Now consider the tokens in H_D^{k-1}. This set consists only of the accumulated knowledge of the tokens that have traversed D. Because the network is uniform, each token in H_D^{k-1} traversed g links before reaching D. Since such token reached D before time t, it reached the former node before time $t-c_1$ and by the induction hypothesis such token enter the network not later than time $(t-c_1)-(g-1)c_1 = t-g\cdot c_1$. □

The next result combines the lemmas above:

Lemma 3.3 Let N be a uniform counting network of depth h with w exits. If the token T is the a^{th} token to exit through the output Y_i at time t then at least $w(a-1)+i+1$ tokens have entered the network by the time $t-h\cdot c_1$.

Proof: Let $e_j = \langle T, Y_i\rangle$. Lemma 3.1 establishes $|H_T^j| \geq w(a-1)+i+1$. Lemma 3.2 establishes that the tokens in $H_T^j = H_{Y_i}^j$ have entered the network no later than the time $t-h\cdot c_1$. □

In the next lemma we show that if there exists time t such that if the tokens in the set K_1 enter the network by the time t and the tokens in the set K_2 enter after t, then any tokens that entered after t can only increase the number of tokens that exit on any output of any node.

Lemma 3.4 Let t by a time instant and $S_1 = \langle K_1, L_1, Q_1\rangle$ and $S_2 = \langle K_1\cup K_2, L_2, Q_2\rangle$ be two timing schedules for a uniform counting network N, such that $K_1\cap K_2 = \emptyset$, $L_1\subseteq L_2$, $Q_1\subseteq Q_2$ and $Q_2(i,1)\leq t < Q_2(j,1)$ for all tokens $i\in K_1, j\in K_2$. If D is a balancing node within the layer $(g+1)$ of N, then at time $t+g\cdot c_2$ the number of tokens that have traversed each of D's outputs in S_2 is no smaller than the number of tokens that have traversed each of D's outputs in S_1.

Proof: By induction on g: For $g=0$ the lemma follows trivially from the fact that in S_1 and S_2, at time t only the tokens in K_1 have entered and they entered through the same input nodes.

Now, assuming the lemma holds for g, we show it holds for $g+1$. Consider a balancer D within the layer $g+2$. Since N is uniform, all of D's inputs are connected to the outputs of some balancers within the layer $g+1$. By the induction hypothesis, at time $t+gc_2$ the number of tokens that traversed these outputs in S_2 is no smaller than in S_1. Thus at time $t+(g+1)c_2$ the number of tokens that enter D in S_2 is no smaller than in S_1.

In any execution, the number of tokens that are output on each output by any balancing node is non-decreasing in time. Since $Q_1\subseteq Q_2$, for any balancing node between the time $t+gc_2$ and $t+(g+1)c_2$ there are at least as many tokens transitioning from its inputs to its outputs in S_2 as in S_1. Since D is a balancer, any additional tokens that enter it will only increase the number of tokens that have exited on its outputs in establishing the output step property. So the lemma follows. □

Lemma 3.5 Let N be a uniform counting network of depth h with w exits. If m tokens enter N by the time t, then by the time $t+h\cdot c_2$ the number of tokens that exit on each output Y_k $(0\leq k<w)$ is at least a_k, where a_k values are uniquely determined by $m=\sum_{i=0}^{w-1} a_i$ and $0\leq a_i - a_j\leq 1$ for any $i<j$.

Proof: Let $S_1 = \langle K_1, L_1, Q_1\rangle$ be a timing schedule with $|K_1| = m$ and $Q_1(k,1)\leq t$ for $k\in K_1$. It takes at most $h\cdot c_2$ time for a token to traverse the network. Therefore, any of the m tokens that enter the network by the time t must exit the network by the time $t_2 = t+h\cdot c_2$. Since no other tokens enter the network by the definition of S_1, then $m=\sum_{i=0}^{w-1} a_i$ and $0\leq a_i-a_j\leq 1$ for any $i<j$ is established, since the counting network is now in a quiescent state.

Suppose additional tokens enter the network after time t. Let S_2 be the timing schedule as in Lemma 3.4 that describes an execution with additional tokens entering after time t. By Lemma 3.4 with $g=h$, for each output Y_k, the new number of tokens that exited in S_2 is no smaller than the number a_k that exited in S_1. □

Now we prove the main theorem about the linearizability of uniform counting networks.

Theorem 3.6 If tokens T_1 and T_2 traverse a uniform counting network during periods $[t_0,t_1]$ and $[t_2,t_3]$ respectively such that $t_1+h\cdot c_2 - 2\cdot h\cdot c_1 < t_2$, then T_2 returns a higher number than T_1.

Proof: Let h be the depth and w be the output width of the network. If a_i is the number of tokens that exited on output Y_i for $0\leq i<w$, we define r as follows:

$$r = \max\{i : 0\leq i<w \wedge a_i = \max\{a_j : 0\leq j<w\}\}$$

By Lemma 3.3, there are at least $m = w(a_r-1)+r+1$ tokens that entered the network no later than the time $t = t_1 - h\cdot c_1$ (see Figure 3).

By Lemma 3.5, at time $t' = t+h\cdot c_2 = t_1 - h\cdot c_1 + h\cdot c_2$, for each output Y_k $(0\leq k<w)$ the number of tokens that exited is at least a_k such that $m=\sum_{i=0}^{w-1} a_i$ and $0\leq a_i-a_j\leq 1$ for any $i<j$. Let K be the set of such tokens.

From the fact that it takes at least $h\cdot c_1$ to traverse the network and because $t_1+h\cdot c_2-2\cdot h\cdot c_1 < t_2$, token T_2 exits at time $t_3\geq t_2+h\cdot c_1 > t_1+h\cdot c_2-2\cdot h\cdot c_1+h\cdot c_1 = t_1+h\cdot c_2-h\cdot c_1$. This means that all tokens that enter

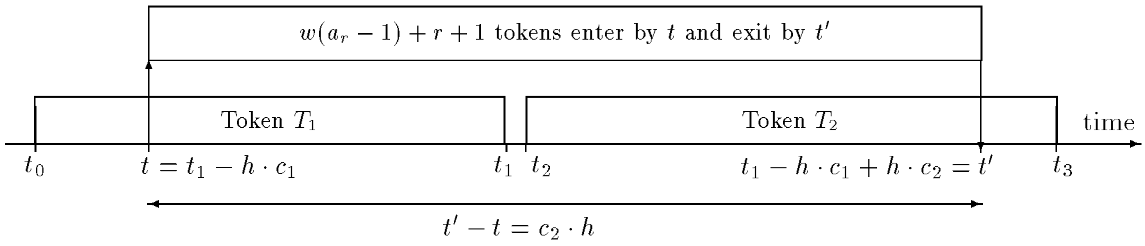


Figure 3: Illustration for Theorem 3.6.

by time $t_1 - h \cdot c_1$ exit before time t_3. Thus, all of the $|K| \geq m$ tokens have exited prior to the exit of token T_2. Since by time t_3 the number of tokens that exit each of the outputs exceeds the number of takens needed to establish the step property using m tokens, token T_2 will return a higher number than any of the m tokens and therefore higher than T_1. □

From the finish-start token time relationship in the above theorem we can establish the following result about the start-start time relationship:

Lemma 3.7 If tokens T_1 and T_2 traverse a uniform counting network during periods $[t_0, t_1]$ and $[t_2, t_3]$ respectively such that $t_0 + 2 \cdot h(c_2 - c_1) < t_2$, then T_2 returns a higher number than T_1.

Note that this corollary is tight. In the full paper we show that if $t_0 + 2 \cdot h(c_2 - c_1) - \varepsilon < t_2$, then for any $\varepsilon > 0$ there is an execution scenario such that T_2 returns a smaller number.

The next corollary also follows from Theorem 3.6 when $c_2 \leq 2c_1$:

Corollary 3.8 If tokens T_1 and T_2 traverse a uniform counting network during disjoint successive time periods $[t_0, t_1]$ and $[t_2, t_3]$ respectively (i.e., $t_1 < t_2$), and $c_2 \leq 2c_1$ then T_2 will return a larger number than T_1.

Together with the definition of linearizability, this leads to the result for uniform networks:

Corollary 3.9 Uniform counting networks are linearizable for $c_2 \leq 2 \cdot c_1$.

The next two corollaries instantiate the linearizability result for specific network definitions.

Corollary 3.10 Bitonic counting networks [4], periodic counting networks [4], the networks of [14] and [7] are linearizable for $c_2 \leq 2 \cdot c_1$.

Corollary 3.11 Counting (diffracting) trees [21] and the uniform trees [8] are linearizable for $c_2 \leq 2 \cdot c_1$.

We now consider the case of $c_2 < k \cdot c_1$ for $k \geq 2$.

Corollary 3.12 For any uniform counting network of depth h and a known constant $k \geq 2$ such that $c_2 < k \cdot c_1$, there exists a linearizable uniform counting network of depth $h \cdot (k-1)$

Proof: Given the original network, we precede each of its inputs with a path of length $h \cdot (k-2)$ of 1-input 1-output "balancers". The tokens traversing such nodes simply proceed to the next balancer. If two tokens traverse the new network in a time-disjoint fashion, then their traversals of the original (sub)network are such that the second token enters it $h \cdot c_2 - 2 \cdot h \cdot c_1$ time after the first token exited. By Theorem 3.6, the second token will return a higher number. □

4 Limits on linearizability of trees and bitonic counters

We now show some limitations on the linearizability of diffracting trees and bitonic networks by constructing execution scenarios that exhibit non-linearizable behavior.

Theorem 4.1 Counting (diffracting) trees are not linearizable if $c_2 > 2 \cdot c_1$.

Proof: Let h be the depth of the tree and let $c_2 = (2+\varepsilon) \cdot c_1$ for some $\varepsilon > 0$. We consider an execution scenario in which the first two tokens enter the tree at the same time t_0. Without loss of generality, let T_0 and T_1 be these tokens that go right (toggle bit transition from 0 to 1 at the root) and left (toggle bit transition from 1 back to 0) respectively. Upon traversing the root, T_0 proceeds at the slowest possible pace, while T_1 proceeds at the fastest possible pace. T_1 reaches the rightmost leaf of the left subtree at time $t_1 = t_0 + h \cdot c_1$ and returns the value 1 (by the definition of the counting tree and c_1).

Immediately after T_1's exit, a wave of $2^h - 1$ tokens enters the tree, say at time $t_2 = t_1 + \delta > t_1$. We choose δ

to be such that $0 < \delta < \varepsilon$. These tokens proceed at the fastest possible pace. Of these tokens, $2^{h-1} - 1$ tokens go to the left subtree and the remaining 2^{h-1} tokens go to the right subtree.

Since the token T_0 is slow, it reaches a leaf at time $t_4 = t_0 + h \cdot c_2$. The second wave fast tokens reaches the leaves at time $t_3 = t_2+h\cdot c_1 = t_1+\delta+h\cdot c_1 = t_0+2\cdot h\cdot c_1+\delta = t_0+h\cdot(c_2-c_1\varepsilon)+\delta = t_0+h\cdot c_2-c_1 h\varepsilon+\delta < t_0+h\cdot c_2$. Thus $t_3 < t_4$ and these fast tokens reach the leaves ahead of T_0. Since we have 2^{h-1} tokens in addition to T_0 traversing the right subtree, at least one token reaches the rightmost leaf of the tree and returns the value 0 (by the quiescence requirement). By the construction, this token traverses the counting tree completely after T_1 exits, but returned a smaller value. □

We now consider bitonic networks.

Lemma 4.2 Given a bitonic counting network, let T_0 be the first token to enter the network through input x_0 and completely traverse the network alone. (a) If T_1 and T_2 are the next two tokens to enter through the same input in this order, then the balancer that is attached to that entrance is the only balancer that both T_1 and T_2 pass through. (b) Token T_0 exits through output y_0, T_1 through output y_1 and T_2 through output y_2 (mod w).

Proof: By induction on the width of the network w: Base case is trivial for $w = 2$ with a single balancer and two counters (we only need to note that outputs y_0 and y_2 are the same for this network).

Assuming the lemma holds for some $w \geq 2$, we now show it holds for networks of width $2w$. The inductive step is depicted in Figure 4, and the node and exit labels below refer to the figure. We use the inductive construction of bitonic counting networks as in [4]. Bitonic[$2w$] is made of two Bitonic[w] networks, two Merger[w] merging networks and additional w balancers. Even-numbered outputs of Bitonic$_1$[w] are connected to the first $w/2$ inputs of Merger$_1$[w] and odd-numbered outputs of Bitonic$_2$[w] are connected to the last $w/2$ inputs of Merger$_1$[w]. The rest of the outputs are similarly connected to Merger$_2$[w]. The outputs of the two mergers are then *shuffled* into a row of w balancers whose outputs are the outputs of Bitonic[$2w$].

Using the inductive hypothesis for Bitonic$_1$[w], T_0 exits via output $u_{1,0}$, T_1 via $u_{1,1}$ and T_2 via $u_{1,2}$ (note that for $w = 2$ the outputs $u_{1,0}$ and $u_{1,2}$ are one and the same). By the construction of Bitonic[$2w$], T_0 and T_1 will enter Merger$_1$[w] via its first balancer. Since these are the only two tokens to enter Merger$_1$[w] and since they traverse the merger one after the other, T_0 must exit via $v_{1,0}$ and T_1 via $v_{1,1}$, else Bitonic[$2w$] will not be in a quiescent state in the execution where T_0 is the only token. Similarly, T_2 will exit via $v_{2,0}$ of Merger$_2$[w]. In the final row of balancers, T_0 and T_2 traverse D_1, and T_1 traverses D_2.

To show (a), we observe that T_0 traverses the network alone and it will reach D_1 first and exit via y_0, and so T_1 will necessarily exit via y_1. The only remaining token T_2 will exit via y_2.

To show (b) we observe that T_1 and T_2 may only traverse the same balancer inside Bitonic$_1$[w], and by the inductive hypothesis, D_0 is the only such balancer. □

Theorem 4.3 Bitonic counting networks are not linearizable if $c_2 > 2 \cdot c_1$.

Proof: In the example in Section 1 we have established that a network of width 2 consisting of single balancer and two counters is not linearizable and it is easy to see that it is not linearizable under the above condition. Below we consider networks with $w > 2$. Let $c_2 = 2\cdot c_1+\varepsilon$ for some $\varepsilon > 0$. Using the framework of Lemma 4.2, we deploy the three tokens T_0, T_1, and T_2 in the following scenario, where $y_0, y_1, \ldots, y_{w-1}$ are the network outputs and w is its width. Starting in the initial state, we let T_0 enter via the input x_0 and completely traverse the network and exit via the output y_0 thus returning the value 0. Following this, token T_1 also enters via x_0 at time t_1, and T_2 enters via x_0 immediately behind T_1 at time $t_1 + \delta_1$ for some $\delta_1 > 0$. We let T_1 proceed at the slowest possible pace, while T_2 proceeds at the fastest possible pace. This means that T_1 exits at time $t'_1 = t_1 + 2h\cdot c_1 + h\varepsilon$, and T_2 exits at time $t'_2 = t_1 + \delta_1 + h\cdot c_1$.

By Lemma 4.2, the paths that T_1 and T_2 traverse have no balancers in common, with the exception of the first balancer in their paths. Thus, in the execution fragment that follows and does not include these tokens' traversal of the first balancer, T_1 is not *influenced* by T_2 and still proceeds to the exit y_1.

As soon as the fast token T_2 exits via y_2 and obtains the counter value 2, w fast tokens enter the network at time $t_3 = t'_2 + \delta_2$ for some $\delta_2 > 0$. Regardless of the paths these token take, they exit the network at time $t'_3 = t_3 + h\cdot c_1$. Since we can choose $\delta_1 + \delta_2 < \varepsilon$, these tokens exit before the slow token T_2.

During this execution, the network is traversed by the total of $w + 3$ tokens. If no other tokens enter the network, then by the quiescence requirement, outputs y_0, y_1, and y_2 have each two tokens that exit through them, and outputs $y_4, \ldots, y_{w-1}$ each have one. Thus one of the fast tokens exits via y_1 and because it is faster than T_1, it obtains the counter value 1, while T_1 obtains the value $1 + w$. As a result the fast token obtains a lower value than T_2. □

As we will see in the experimental section that follows, the c_2/c_1 ratio greater then 2 may cause higher percentage of non-linearizable operations. In the theorem below

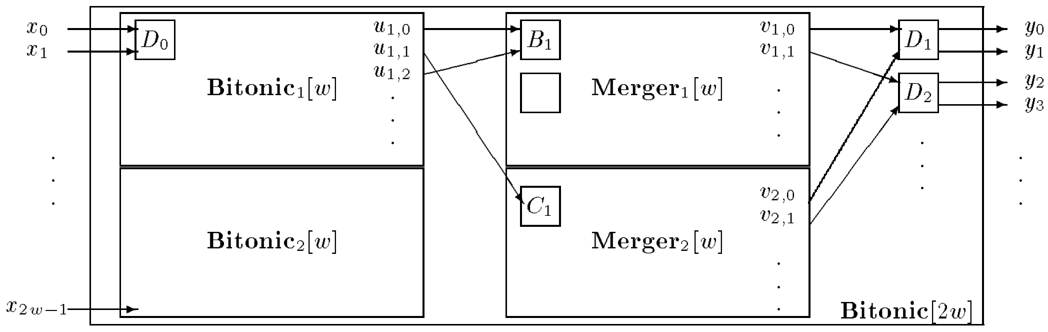


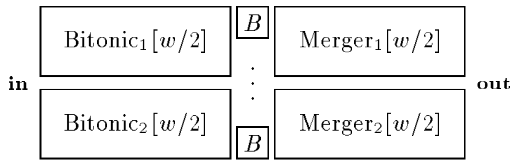
Figure 4: Inductive step for Lemma 4.2.

we show that for bitonic networks there can be a large fraction of tokens that exhibit non-linearizable behavior for certain c_2/c_1:

Theorem 4.4 Bitonic counting networks are not linearizable if $c_2 > \frac{3+\log w}{2} \cdot c_1$, where w is the width of the network.

Proof: Bitonic counting networks [4] of width w, Bitonic$[w]$ have depth $h = \frac{\log w \cdot (\log w + 1)}{2}$. The network consists of two stages, the first stage includes two Bitonic$[w/2]$ networks of depth $h_1 = h - \log w$ connected in parallel to the second stage that is a merging network of depth $h_2 = \log w$. Merger$[w]$.

Merger$[w]$ consists of a row of balancers connected to two Merger$[w/2]$ mergers (for details see [9]). Note that this inductive construction of the merger is different from, but isomorphic to the construction in Figure 4.



A non-linearizable schedule is constructed as follows: The first wave of $w/2$ tokens enters at the same time Bitonic$_1[w/2]$ network and proceed in lock step at some pace to the exits of the first stage. The second wave of $w/2$ tokens enters the same network immediately behind the first wave after a small delay $\delta > 0$.

As soon as the first wave enters Merger$[w]$, it slows down to the slowest possible pace of one link per c_2 time. This wave proceeds to the Merger$_1[w/2]$ sub-component of the merger after passing through the top row of balancers of Merger$[w]$.

Similarly, the second wave proceeds to the Merger$_2[w/2]$ sub-component, except that it proceeds at the fastest possible pace of c_1 time per link. As soon as the second wave exits, a third wave enters Bitonic$[w]$ as the first two waves.

The third wave proceeds in lock step at the fastest possible pace of c_1 time per link to the exits. By the quiescence requirement, this wave exits through the first $w/2$ exits.

It takes the first wave $t_1 > h_2 \cdot c_2 = c_2 \cdot \log w$ time to reach the exits. It takes the second wave $t_2 = h_2 \cdot c_1 = c_1 \cdot \log w$ time to exit. It takes the third wave $t_3 = h \cdot c_1 = c_1 \cdot \frac{\log w \cdot (\log w + 1)}{2}$ time to traverse the entire network. Since $c_2 > \frac{3+\log w}{2} \cdot c_1$, we have that $t_1 > t_2 + t_3$. Thus the third wave passes the first wave on the final link out and returns counter values that are all lower than those obtained by the second wave. □

5 Simulation results

We empirically evaluated the linearizability of counting networks on a simulated 256 processor distributed-shared-memory machine similar to the MIT *Alewife* machine [2] of Agarwal et al. Our simulations were performed using *Proteus* , a multiprocessor simulator developed by Brewer, Dellarocas, Colbrook and Weihl [6]. In our benchmark a certain fraction $F = 25\%, 50\%$ of the processors waits $W = 100, 1000, 10000, 100000$ cycles after traversing a node in the net. The execution is stopped when 5000 operations were performed. The data collected is the non-linearizability ratio, i.e the percentage of non-linearizable operations (see Definition 2.4) among all the operations during the execution. The networks implemented are the diffracting tree [21] and the bitonic counting network [4]. Both the network and the tree are of width 32.

Every balancer is implemented as a critical section protected by a Mellor-Crummey and Scott (MCS) queue-lock [18] and, in the diffracting tree, using the multi-prism implementation of [20]. This is done to reduce contention on the nodes which would have attenuated the influence of the W-waiting periods on the $c2/c1$ relation. The results are given in Figures 5 and 6. The y-axis shows the non-linearizability ratio.

In the table in Figure 7 we provide the average c_2/c_1 ratio as measured during the simulation. The average c_2/c_1 is defined as *(Tog+ W)/Tog* where *Tog* is the average time a token waits before toggling the balancer (clearly one can add other factors, e.g., standard deviation, to the analysis, which we defer to the full version for lack of space). As can be seen, for the lower delay levels the average c_2/c_1 ratio is less then or around 2, and no violations were detected. When the average goes up far above 2, violations occur. Diffracting trees have a higher fraction of violations because of their lower depth, which means that there is less of a padding effect as implied by Theorem 3.6.

Notice that the number of violations increases with concurrency. As more processors pass through the network concurrently, the slow processes have fewer transitions that affect fast transitions. The $100,000$ becomes better than the $10,000$ at high concurrency levels because each delayed token is effectively standing still so the total number of such slow transitions while fast tokens traverse is much smaller than in the $10,000$ case.

We also tested the linearizability of these implementation when $F = 0\%, 100\%$ and/or $W = 0$ and no non-linearizable operations were detected. Another scenario in which every token waits a random number of cycles between 0 and W was also simulated and was observed to be completely linearizable.

This experimental study shows that there are counting network implementations that over a wide range of timing variations remain linearizable.

References

[1] E. Aharonson and H. Attiya. Counting networks with arbitrary fan out. In *Proceedings of the* 3^{rd} *Symposium on Discrete Algorithms,* Orlando, Florida, January 1992. Also: Technical Report 679, The Technion, June 1991.

[2] A. Agarwal, D. Chaiken, K. Johnson, D. Krantz, J. Kubiatowicz, K. Kurihara, B. Lim, G. Maa, and D. Nussbaum. The MIT Alewife Machine: A Large-Scale Distributed-Memory Multiprocessor. To appear in *Scalable Shared Memory Multiprocessors*, Kluwer Academic Publishers, 1991. Also as MIT Technical Report MIT/LCS/TM-454, June 1991.

[3] B. Aiello, R. Venkatesan, and M. Yung. Coins, Weights and Contention in Balancing Networks. In *Thirteenth ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing,* August 1994, pp. 193-214.

[4] J. Aspnes, M.P. Herlihy, and N. Shavit. Counting Networks and Multi-Processor Coordination. In *Proceedings of the 23rd Annual Symposium on Theory of Computing,* May 1991.

[5] E.A. Brewer, C.N. Dellarocas. PROTEUS *User Documentation.* MIT, 545 Technology Square, Cambridge, MA 02139, 0.5 edition, December 1992.

[6] E.A. Brewer, C.N. Dellarocas, A. Colbrook and W.E. Weihl. PROTEUS: A High-Performance Parallel-Architecture Simulator. MIT Technical Report /MIT/LCS/TR-561, September 1991.

[7] C. Busch and M. Mavronicolas. A Combinatorial Treatment of Balancing Networks. In *13th ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing,* August 1994, pp. 206–215.

[8] C. Busch and M. Mavronicolas. New Bounds on Depth and Contention for Counting Networks. preprint, Univ. of Cyprus, October 1995.

[9] T.H. Cormen, C.E. Leiserson, and R. L. Rivest. Introduction to Algorithms MIT Press, Cambridge MA, 1990.

[10] E.W. Felten, A. LaMarca, R. Ladner. Building Counting Networks from Larger Balancers. University of Washington T.R. #93-04-09.

[11] A. Gottlieb, B.D. Lubachevsky, and L. Rudolph. Basic techniques for the efficient coordination of very large numbers of cooperating sequential processors. *ACM Transactions on Programming Languages and Systems,* 5(2):164–189, April 1983.

[12] M.P. Herlihy, N. Shavit, and O. Waarts. Linearizable Counting Networks. In *Proceedings of the* 32^{nd} *Annual Symposium on Foundations of Computer Science,* San Juan, Puerto Rico, October 1991, pp. 526-535. Detailed version with empirical results appeared as MIT/LCS technical manuscript 459, November 1991.

[13] M.P. Herlihy and J.M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems,* 12(3):463–492, July 1990.

[14] M. Klugerman and C.G. Plaxton. Small-depth Counting Networks. In *ACM Symposium on Theory of Computing (STOC),* 1992

[15] M. Klugerman, Small-Depth Counting Networks. Ph.D. Thesis, MIT, 1994.

[16] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers,* C-28(9), September 1979

[17] N.A. Lynch and M.R. Tuttle. Hierarchical Correctness Proofs for Distributed Algorithms. In *Sixth ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing,* August 1987, pp. 137–151. Full version available as MIT/LCS/TR–387.

[18] J.M. Mellor-Crummey and M.L. Scott. Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors. Technical Report 342, University of Rochester, Rochester, NY 14627, April 1990.

[19] C.H. Papadimitriou. The serializability of concurrent database updates. *Journal of the ACM,* 26(4):631-653, October 1979.

[20] N. Shavit, and D. Touitou. Elimination Trees and the Construction of Pools and Stacks. In *Proceedings of the Annual Symposium on Parallel Algorithms and Architectures (SPAA),* June 1995.

[21] N. Shavit and A. Zemach. Diffracting Trees. In *Proceedings of the Annual Symposium on Parallel Algorithms and Architectures (SPAA),* June 1994.

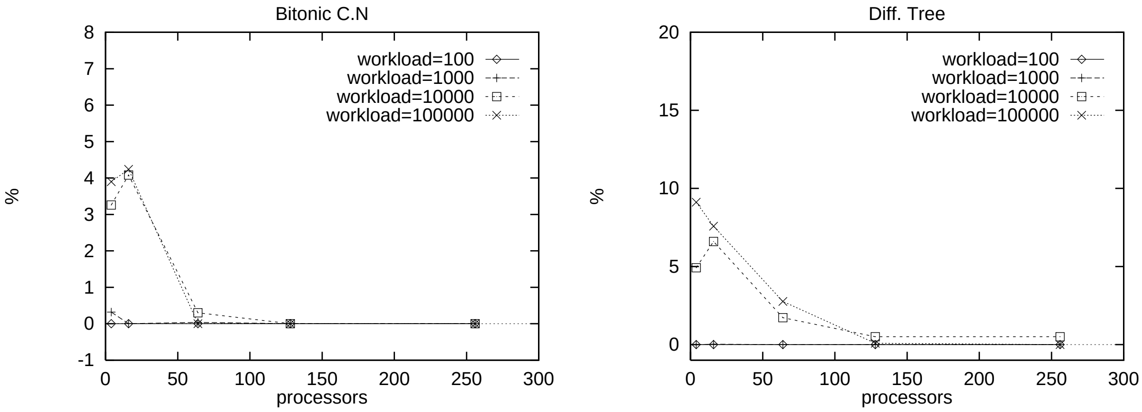


Figure 5: Non-linearizability Ratios for $F = 25\%$ delayed processors.

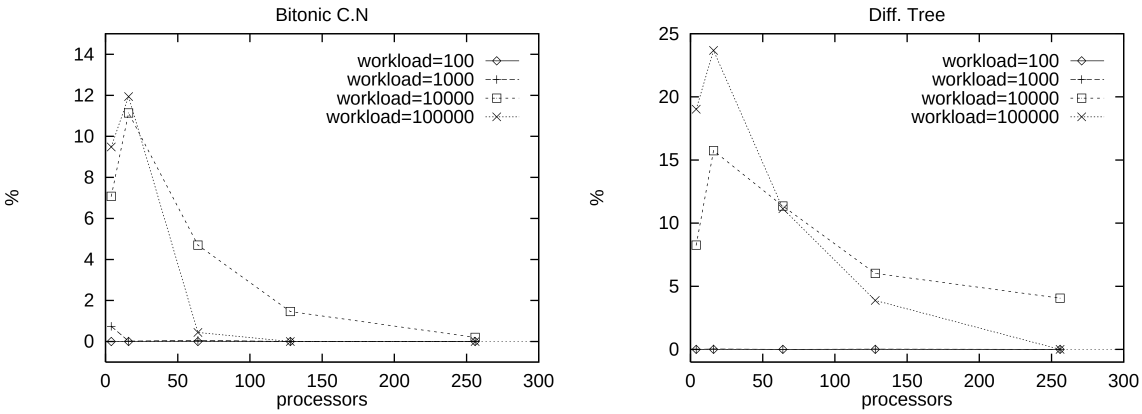


Figure 6: Non-linearizability Ratios for $F = 50\%$ delayed processors.

	Bitonic Counting Network					Diffracting Tree				
Workload	n=4	n=16	n=64	128	n=256	n=4	n=16	n=64	128	n=256
50 %										
100	1.45	1.39	1.25	1.22	1.18	1.11	1.11	1.10	1.11	1.11
1000	5.67	5.03	3.70	3.24	2.73	2.06	2.06	1.94	2.01	2.09
10000	48.77	41.26	27.98	24.49	21.21	12.14	11.55	10.10	10.57	11.36
100000	483	410.21	280.27	244.34	215.22	115.54	107.39	91.86	96.72	105.62
25 %										
100	1.45	1.39	1.25	1.22	1.17	1.11	1.11	1.10	1.11	1.11
1000	5.54	4.95	3.56	3.16	2.68	2.06	2.08	1.96	2.03	2.09
10000	46.18	40.15	26.67	23.39	19.63	11.67	11.70	10.38	10.97	11.78
100000	456.70	395.70	262.08	226.80	193.06	108.42	107.96	93.89	101.02	109.12

Figure 7: Average c_2/c_1 in the simulations of bitonic networks and diffracting trees.