Timing Conditions for Linearizability in Uniform Counting Networks

Nancy Lynch, *MIT*
and
Nir Shavit, *MIT and Tel-Aviv University*
and
Alex Shvartsman, *University of Connecticut*
and
Dan Touitou, *Tel-Aviv University and nSOF Israel*

Counting networks are concurrent data structures that serve as building blocks in 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. *Linearizability* is an important consistency condition for concurrent data structures, as it simplifies proofs and enhances compositionality.

Though most counting networks are not linearizable, this paper presents a precise characterization of the timing conditions under which uniform non-linearizable networks exhibit linearizable behavior. Uniformity is a common structuring property of almost all published counting networks: a uniform network is made of "balancers" and "wires" so that each balancer lies on some path from inputs to outputs, and all paths from inputs to outputs have equal lengths. Our results include the following simple condition: if the time it takes a slow token to traverse a "wire" or "balancer" is no more than twice that of a fast token, the network is linearizable. Surprisingly, the timing measure in this condition is *local* to the individual "wires" and "balancers" of the network, that is, it is independent of network depth.

We use our timing measure to mathematically explain our empirical findings: that in a variety of highly concurrent execution scenarios tested on a simulated shared memory multiprocessor, the *Bitonic* counting networks of Aspnes, Herlihy, and Shavit exhibit completely linearizable behavior, and when linearizability is violated, the percentage of violations is relatively small.

Herlihy, Shavit, and Waarts have shown that counting networks that achieve linearizability under all circumstances must pay the penalty of linear time latency. Our results suggest that for systems in which timing anomalies occur infrequently, such linear delays may be an unnecessary burden on applications that are willing to incur occasional non-linearizability.

This work was supported by the following contracts: ARPA N00014-92-J-4033 and F19628-95-C-0118, NSF 922124-CCR and 9520298-CCR, and ONR-AFOSR F49620-94-1-01997. A preliminary version of this work appears as *Counting Networks are Practically Linearizable* in the Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, pages 280-289, Philadelphia, PA, May 1996. Contact author: A. Shvartsman. Department of Computer Science and Engineering, 191 Auditorium Road, U-155, University of Connecticut, Storrs, CT 06269. Email: `aas@cse.uconn.edu`.

1. INTRODUCTION

Counting networks [4] are a class of highly scalable structures used for concurrent counting. Such networks allow the design of concurrent data structures in a way that eliminates sequential bottlenecks and contention. Unlike queue-locks [21] and combining trees [13] which are based on a single counter location handing out indices, counting networks hand out indices from a collection of counter locations. To guarantee that indices handed out by the separate counters are not erroneously "duplicated" or "omitted," one adds a special network coordination structure to be traversed by processes before accessing the counters.

Counting networks [4] are constructed from simple computing elements called *balancers* (see Figure 1). Tokens arrive on the balancer's input wires and are output on its output wires. Intuitively one may think of a balancer as a toggle mechanism that, given a stream of input tokens, repeatedly sends one token to the left output wire and one to the right, effectively balancing the number of tokens that have been output. In order to form a counting network, balancers are connected to one another by wires in an acyclic fashion, in the same way comparators are connected to form a sorting network [11]. However, unlike in sorting networks, counting networks are asynchronous in nature, that is, tokens arrive at the network's input wires at arbitrary times, and traverse the network with differing pace. Nevertheless, if the balancers are connected correctly, a network having w consecutively numbered output wires will move input tokens to output wires in increasing order modulo w. Networks of balancers having this property can easily be adapted to count the total number of tokens that pass through them. Counting is done by adding a "local counter" to each output wire i, so that tokens coming out of that wire are assigned numbers $i, i+w, i+2w$, and so on.

On a shared memory multiprocessor, counting networks are implemented as data structures in which balancers are represented as records and wires as pointers among them. Tokens are "shepherded" by processors that traverse this pointer-based data structure from input pointers to output wires, finally incrementing the counter on the appropriate output wire. This implies that tokens may overtake one another on a wire and that balancer and network traversal times are dependent on individual processor speeds and variations in speeds.

A Bitonic counting network [4] has a layout isomorphic to Batcher's Bitonic sorting network [7]. Bitonic counting networks for n processors have width $w < n$ and depth $\Theta(\log^2 w)$ (all logarithms in this paper are to the base 2). Unlike combining trees, counting networks support complete independence among requests and are thus highly fault tolerant. At peak performance their throughput is w, as w indices are returned per time step by the independent counters. Unfortunately, counting networks suffer a performance drop-off due to contention as concurrency increases, and the latency in traversing them is a high $\Theta(\log^2 w)$. There is a wide body of research on counting networks [2; 3; 4; 9; 10; 12; 15; 17; 18]. A recently developed form of counting network called a Diffracting Tree [24] is based on a new type of distributed balancer implementation. It has been shown to scale especially well, exhibiting low latency since its depth is logarithmic in w.

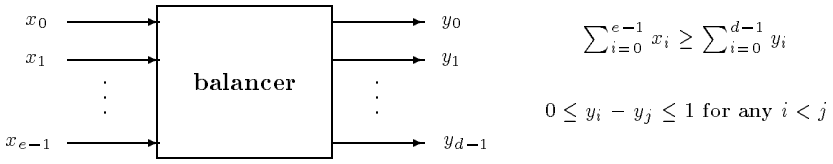

Fig. 1. A balancer and its input-output properties.

Linearizability is a consistency condition for concurrent systems formulated by Herlihy and Wing [16]. It requires that the values returned by access requests to a concurrent shared object reflect the order in which they were issued. The use of linearizable data abstractions simplifies both the specification and the proofs of multiple instruction/multiple data shared memory algorithms. As Herlihy and Wing explain, linearizability generalizes and unifies a number of *ad-hoc* correctness conditions in the literature, and is related to (but not identical with) correctness criteria such as sequential consistency [19] and strict serializability [22].

Herlihy, Shavit, and Waarts [15] defined the class of linearizable counting networks, 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 [24], there exist worst case asynchronous schedules in which linearizability is violated. In [15] linear depth linearizable counting network constructions were presented and shown to be optimal, that is, any low contention counting network that is linearizable in all executions must have linear depth.

Timing and Linearizability

This paper provides a characterization of the timing conditions under which low depth non-linearizable counting networks become linearizable. It applies to semi-synchronous and real-time systems [6] where upper and lower time bounds that limit the extent to which one process can be slower or faster than others are known. As we show, our characterization also extends beyond such systems and has implications in the analysis of counting network linearizability in general asynchronous multiprocessor systems. We believe that the linear time cost of designing counting networks achieving linearizability under all circumstances may be an unnecessary burden on applications that are willing to trade-off occasional non-linearizability for speed and parallelism. In such systems an intelligent trade-off decision can be made with the help of clear characterization of the parameters governing linearizability.

Our main result is a simple timing condition that is *local* to the individual wires and balancers of the network. It quantifies the extent to which a network can suffer from timing anomalies and still remain linearizable.

This result is interesting, since even a counting network of depth one exhibits

non-linearizable behavior. Consider the following 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.

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.

We use a c_1/c_2 timing model in the style of Attiya, Dwork, Lynch, and Stockmeyer [5]. 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 standard message passing and shared memory balancer implementations [4; 24]. Alternately, one could attribute the c_2/c_1 latency to the balancer traversal and make wire traversal instantaneous. The two models can be shown to be equivalent, and we choose to attribute delays exclusively to the wires as this simplifies our modeling and presentation.

Our model is also similar to that of semi-synchronous systems (cf. Archimedean distributed systems of Vitanyi [25]). One can view our setting as one in which each token traverses a wire and a balancer on the local clock tick, where the local clocks can tick not faster than every c_1, and not slower than every c_2 time units according to some global clock.

A common structuring property of almost all published counting networks [2; 4; 3; 9; 12; 15; 17; 18; 23; 24] is *uniformity*: each balancer of the network lies on some path from inputs to outputs, and all paths from inputs to outputs have equal lengths.

We prove, in Section 3, the following properties for any uniform counting network (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 balancers so that the resulting network is a linearizable network of depth $O(h)$.

In Section 4 we show that counting (Diffracting) trees and Bitonic counting networks are not linearizable for $c_2 > 2 \cdot c_1$, and that one can create executions with large numbers of non-linearizable operations.

Finally, in Section 5 we provide empirical measurements of the extent to which

timing can affect linearizability in Bitonic Networks and Diffracting Trees. These results were collected on a simulated Alewife [1] shared-memory multiprocessor using the Proteus [8] simulator.

We use our c_1/c_2 measure to mathematically support our experimental results: that in a variety of "normal" situations, the Bitonic counting networks of Aspnes, Herlihy, and Shavit [4] exhibit linearizable behavior. In fact, for high concurrency levels, our results show that even if one skews system timings by introducing large timing variations among processes, the network rarely exhibits violations of linearizability. At low concurrency levels we observed a significantly higher number of violations.

2. MODELS AND DEFINITIONS

We consider networks consisting of acyclically wired routing elements called *balancers*. We refer the reader to [4] for a more detailed presentation of the model and its implications. For the sake of generality, our balancers are defined as multi-balancers in the style of Aharonson and Attiya [2] and Felten, LaMarca, and Ladner [12] (Figure 1), having e input wires $x_0, x_1, \cdots, x_{e-1}$ and d output wires $y_0, y_1, \cdots, y_{d-1}$. Slightly abusing notation, we let x_i (respectively y_i) also serve as a state variable that stands for the number of tokens that have entered (exited) via that wire.

A balancer passes tokens from input wire to output wire, maintaining a *step property* on its output wires: in any state of the balancer, its output wires satisfy $0 \leq y_i - y_j \leq 1$ for any $i < j$. This requirement is stronger than the standard one [4], since it implies that token traversal through a balancer is atomic. However, we note that it is consistent with the standard message passing and shared memory based balancer implementations [4] and with Diffracting balancer implementations [24], as they all meet the specification of a balancer with atomic transitions.

We further require that a balancer not create tokens spontaneously, that is, $\sum_{i=0}^{e-1} x_i \geq \sum_{i=0}^{d-1} y_i$. A state in which $\sum_{i=0}^{e-1} x_i = \sum_{i=0}^{d-1} y_i$ is called a *quiescent* state.

To perform an increment operation on the network, a process routes a token from input wire to output wire, traversing a sequence of balancers on the way. We define a *quiescent* state of a balancing network with v input ports $X_0, X_1, \ldots, X_{v-1}$ and w output ports $Y_0, Y_1, \ldots, Y_{w-1}$ as a state in which all tokens that have ever entered it have already exited. A *counting network* with w outputs is a network of balancers that satisfies the following *step property*:

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

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

We now add timing to our model. The state transition of a balancer, i.e., the passing of a token from the balancer's input port to its output port, will be modeled as an instantaneous event. While balancer transitions are instantaneous, transitions

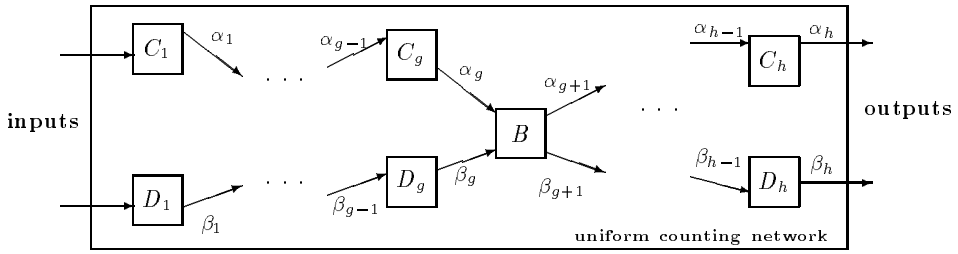

Fig. 2. Equal length paths lead to any balancer in a uniform network.

along a wire connecting an output port of one balancer to an input port of another are not. However, we assume that there is some $c_1 > 0$ that is the *lower bound* on time it takes for a token to traverse a wire between two balancers. Similarly there exists a c_2 that is the *upper bound* on such time, where $0 < c_1 \leq c_2$. Wires with the same delay bounds are also used to connect the output wires of the network to a set of counters added to it. Each output wire Y_i of the network leads from a balancer whose output wire is also a network output, to an atomic counter at its end. We identify this counter with the output wire Y_i. The input wires of a network are the input wires of the balancers they connect to. Such balancers are called *input balancers*. We use the term *node* to refer to a component of a network that may be either a balancer or a counter.

We refer to w as the *output width* of the network. The tokens exiting from output wire 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 balancer 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 wires on the path between any input balancer 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 balancer B, the lengths of all paths from the input balancers to B are equal and the lengths of all paths from B to the output balancers are equal, see Figure 2. Note that there and in the remaining figures, we do not show the counters attached to the outputs. For $1 \leq g \leq (h + 1)$ we also define the g-th *layer* of a network to be the collection of nodes (balancers or counters) whose distance from the inputs is $g - 1$.

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

An execution or execution sequence of a network is a sequence $E = e_1, e_2, \ldots$ of instantaneous transition events $e_i = \langle T, B \rangle$ corresponding to a token T traversing a balancer or counter B. We associate *history variables* with tokens and balancers 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_B with each balancer B. For every execution E the values of these variables are computed inductively as follows, where H_B^i and H_T^i denote the values of H_B and H_T after the event e_i:

—At the beginning of the execution, we define $H_B^0 = \emptyset$ and $H_T^0 = \{T\}$. In other words, at the beginning of the execution the knowledge of every balancer is an empty set and the knowledge of every token consists of the token's own identifier.
—The inductive step is as follows: If $e_i = \langle T, B \rangle$, then $H_B^i = H_T^i = H_T^{i-1} \cup H_B^{i-1}$. Intuitively, the token T and the balancer B combine their knowledge as the result of e_i.
For every other token $T' \neq T$ and balancer $B' \neq B$, we define $H_{T'}^i = H_{T'}^{i-1}$ and $H_{B'}^i = H_{B'}^{i-1}$.

Definition 2.2 A *timing schedule* S for an execution of a uniform network of depth h and input width v is a triple $\langle K, L, Q \rangle$. K is the set of token ids produced by sequentially numbering the tokens starting with 1 and based on their arrival times. $L : K \rightarrow \{X_i : 0 \leq i < v\}$ is a function such that for a token T, $L(T)$ is the input balancer on which the token enters the network. $Q : K \times [1..(h+1)] \rightarrow \mathbf{R}$ (where $\mathbf{R}$ is the reals) is the function such that $Q(T, g)$ is the real time instant when the token T passes through a node in layer g of the network.

Adapting the definition of Herlihy and Wing [16] to counting networks:

Definition 2.3 An execution of a counting network is *linearizable* if for any two tokens that traverse the network one completely after another (non-overlapping in time), the earlier token obtains a smaller value than the later one.

Definition 2.4 A counting network is *linearizable* if every execution of the network is linearizable.

We now introduce the notion of non-linearizable operations. Consider an execution in which the network traversal operation α completely precedes another traversal operation β, but α returns a higher value than β. Clearly such an execution is not linearizable. In the definition below we ascribe the non-linearizablilty of the execution to the operation β:

Definition 2.5 Given an execution of a counting network, we say that a traversal operation β and its associated token are *non-linearizable*, if there exists some other traversal operation α completely preceding β in time, whose associated token has a higher returned value than β.

We choose to define β as the non-linearizable operation and not α since this allows us to determine whether or not an operation is non-linearizable as soon as it completes. Furthermore, if instead α were defined to be the non-linearizable traversal operation, this would lead to non-intuitive situations where a single operation can cause all preceding operations to become non-linearizable if it returns a sufficiently low value.

It is easy to see that for any execution sequence, if we remove all non-linearizable traversal operations the remaining sequence of operations will contain no violations of linearizability[1]. However, such sequence of operations might not correspond to a valid execution of a counting network, since it could contain gaps.

The following definition quantifies non-linearizability of finite executions:

Definition 2.6 The *fraction* of non-linearizable operations in a finite execution is defined to be the number of non-linearizable operations divided by the number of completed operations in the execution.

It follows from the definitions above that this fraction is an upper bound on the fraction of operations whose removal yields a linearizable execution trace.

3. A CHARACTERIZATION OF LINEARIZABILITY FOR COUNTING NETWORKS

In this section and the next, we show that the ratio c_2/c_1 plays a key role in determining whether a uniform counting network is linearizable.

We begin by proving several lemmas that will be used to derive our main result, that uniform networks are linearizable for $c_2 \leq 2c_1$. The first lemma shows that in any counting network, when a token completed traversing 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 $|H_T| \geq w(a-1)+i+1$ following its transition onto Y_i.

Proof: The proof is 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 $e_k = \langle T_k, B_k \rangle$ and $e_{k+1} = \langle T_{k+1}, B_{k+1} \rangle$, then either $T_k = T_{k+1}$ or $B_k = B_{k+1}$.

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 projection of E consisting 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 claim that E' is a possible execution of the counting network in which the participating tokens and nodes cannot distinguish between E' and E.

[1] In general it may be possible to remove fewer operations (whether linearizable or not) to eliminate all instances of non-linearizability. For example, consider an execution consisting of three time-disjoint operations α, β and γ that return the values 3, 1 and 2, in that order. According to our definition, β and γ are non-linearizable. Removing both of them yields a sequence consisting of α alone, thus removing all instances of non-linearizability. However, if we remove α instead, then β and γ become linearizable.

We show this by induction on all the prefixes of E'. The base case for the empty prefix is trivial. For the inductive step we assume that the length of E' is positive and that the prefix of E' of length $n-1$, for $n \geq 1$, is a possible execution of the network. We now consider the prefix $e'_1, e'_2 \ldots, e'_n$ of E', where $e'_n = \langle S, D \rangle$.

Now consider the sequence $e_1, e_2, \ldots, e_m$ such that it is the prefix of E that ends with $e_m = e'_n$. By the definition of E', we know that all the events involving either S or D in $e_1, e_2, \ldots e_{m-1}$ are contained in $e'_1, e'_2, \ldots, e'_{n-1}$. By the induction hypothesis, $e'_1, e'_2, \ldots, e'_{n-1}$ is a possible execution of the counting network in which the participating tokens and nodes cannot distinguish between this prefix and the prefix $e_1, e_2, \ldots e_i$ of E, where the event e_i is e'_{n-1}.

Note that by the definition of E' the subsequence $e_{i+1}, \ldots, e_{m-1}$ of E, does not include any events involving S or D. Therefore, neither S nor D can distinguish between the execution $e'_1, e'_2, \ldots e'_{n-1}$ and the execution $e_1, e_2, \ldots e_{m-1}$. Because $\langle S, D \rangle$ is next event after e_{m-1} in E, the sequence $e'_1, e'_2, \ldots e'_{n-1}, \langle S, D \rangle$ 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', any completion of E' in which no new token enters the network leads to a quiescent state with the step property violated. This is so because if a tokens exit on Y_i, then it is impossible to establish the needed step property with fewer than $w(a-1)+i+1$ tokens. □

The next lemma shows that the implicit knowledge in the history variables can only reflect information propagation at the maximum pace of 1 wire per c_1 time units.

Lemma 3.2 Let N be a uniform counting network of depth h. For any execution $E = e_1, e_2, \ldots$, if $e_k = \langle T, B \rangle$ occurs at time t, where B is a node in layer $(g+1)$, for $0 \leq g \leq h$ then H_B^k contains only tokens that enter the network by time $t - g \cdot c_1$.

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

Assume there is an execution sequence $E = e_1, e_2, \ldots e_k, \ldots$, containing a transition event $e_k = \langle T, B \rangle$ that occurs at time t. Assume also that $|\{e_j : 1 \leq j < k \wedge e_j = \langle T, B_j \rangle\}| = g$, which means that token T traverses g balancers and wires *en route* to B. From the definition of historical knowledge, $H_T^k = H_T^{k-1} \cup H_B^{k-1}$.

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

Now consider the tokens in H_B^{k-1}. This set consists of the accumulated knowledge of the tokens that traversed B. Because the network is uniform, each token in H_B^{k-1} traverses g wires before reaching B. Since each such token reaches B by time t, it reaches the previous balancer (there is such a balancer because $g > 0$) by time $t - c_1$ and by the induction hypothesis it enters the network by 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 outputs. If at time t, token T exits on output Y_i, and it is the a^{th} token to exit through this output wire, then at least $w(a-1)+i+1$ tokens enter the network by time $t-h\cdot c_1$.

Proof: Let $e_j = \langle T, Y_i \rangle$ (recall that we identify the counter at output Y_i with Y_i). Lemma 3.1 establishes $|H_T^j| \geq w(a-1)+i+1$. Lemma 3.2 establishes that the tokens in $H_T^j = H_{Y_i}^j$ enter the network by time $t - h \cdot c_1$. □

In the next lemma we show that if the tokens in a set K_1 enter a network N by time t and proceed according to time schedule Q_1, and the tokens in the set K_2 enter after t, then any tokens that enter after t can only increase the number of tokens that exit on any output of any balancer B as the result of Q_1.

Lemma 3.4 Let t be 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 $Q_2(T_1, 1) \leq t < Q_2(T_2, 1)$ for all tokens $T_1 \in K_1, T_2 \in K_2$. If B is a balancer within layer $g+1$ of N, where $0 \leq g \leq h$, then by time $t + g \cdot c_2$ the number of tokens that traverse any of B's outputs in S_2 is no smaller than the number of tokens that traverse the same output of B 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, by time t only the tokens in K_1 enter and they enter through the same input balancers.

Assuming the lemma holds for g, we show it holds for $g+1$. Consider a node B within the layer $g+2$. Since N is uniform, all of B's inputs are connected to the outputs of some balancers within the layer $g+1$. By the induction hypothesis, by time $t + gc_2$ the number of tokens that exit on any of these outputs in S_2 is no smaller than the number that exit on the same outputs in S_1. Since it takes at most c_2 time to traverse a wire from one layer to the next, by time $t + (g+1)c_2$ the number of tokens that enter any of the inputs of B in S_2 is no smaller than the number of tokens entering the same inputs in S_1.

In any execution, the number of tokens exiting any of the outputs of a balancer is deterministically established from the sum of the number of tokens that enter the inputs of the balancer. Since $Q_1 \subseteq Q_2$, for any balancer, between time $t + gc_2$ and $t + (g+1)c_2$ there are at least as many tokens transitioning from its inputs to each of its outputs in S_2 as in S_1. □

For the next two proofs, given a counting network of width w, we define q_i^m to be the number of tokens that exit on each of the network outputs Y_i ($0 \leq i < w$) once m tokens enter and exit the network. We use the property of counting networks that q_i^m is uniquely defined by the formulas $\sum_{i=0}^{w-1} q_i^m = m$ and $0 \leq q_i^m - q_j^m \leq 1$ for $i < j$ [4].

Lemma 3.5 Let N be a uniform counting network of depth h and width w. If m tokens enter N by time t, then by time $t + h \cdot c_2$ the number of tokens that exit on

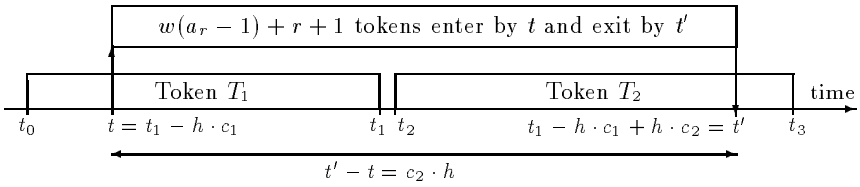

Fig. 3. Illustration for Theorem 3.6.

each output Y_i is at least q_i^m.

Proof: Let $S_1 = \langle K_1, L_1, Q_1 \rangle$ be a timing schedule with $|K_1| = m$ and $Q_1(T, 1) \leq t$ for $T \in K_1$. 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 time t must exit the network by time $t' = t + h \cdot c_2$. Since by the definition of S_1 no other tokens entered the network, it is in a quiescent state and the number of tokens exiting on each output Y_i is exactly q_i^m.

Suppose additional tokens enter the network after time t. Let S_2 be the timing schedule that describes an execution with additional tokens entering after time t. By Lemma 3.4 with $g = h$, for each output Y_i, the new number of tokens that exit in S_2 is no smaller than the number that exit in S_1, and is therefore at least q_i^m. □

The following is our main theorem on the linearizability of uniform counting networks.

Theorem 3.6 If tokens T_1 and T_2 traverse a uniform counting network of depth h during periods $[t_0, t_1]$ and $[t_2, t_3]$ respectively, in an execution in which $t_1 + h \cdot (c_2 - 2c_1) < t_2$, then T_2 has a higher returned value than T_1.

Proof: Suppose a_i is the number of tokens that exit by time t_1 on output Y_i for $0 \leq i < w$. We define r as follows:

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

that is, r is the largest output index such that a_r is the largest number of tokens that exit on any output.

By Lemma 3.3, there are at least $m = w(a_r - 1) + r + 1$ tokens that enter the network no later than time $t = t_1 - h \cdot c_1$ (see Figure 3), and T_1 is among these tokens. Let K be the set of these tokens.

By Lemma 3.5, by time $t' = t + h \cdot c_2 = t_1 - h \cdot c_1 + h \cdot c_2$ the tokens in K exit, and for each output Y_i ($0 \leq i < w$) the number of tokens that exit is at least q_i^m.

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 $t_3 \geq t_2 + h \cdot c_1 > t_1 + h \cdot c_2 - 2 \cdot h \cdot c_1 + h \cdot c_1 = t_1 + h \cdot c_2 - h \cdot c_1 = t'$. This means that all tokens that enter by time $t = t_1 - h \cdot c_1$ exit before time t_3. Thus, all of the tokens in K exit prior to the exit of token T_2. Since by time t_3 the number of tokens that exit each of the outputs Y_i exceeds the

number of tokens q_i^m needed to establish the step property using m tokens, token T_2 returns a higher number than any of the tokens in K 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:

Corollary 3.7 If tokens T_1 and T_2 traverse a uniform counting network of depth h during periods $[t_0, t_1]$ and $[t_2, t_3]$ respectively in an execution where $t_0 + 2h \cdot (c_2 - c_1) < t_2$, then T_2 has a higher returned value than T_1.

Proof: From the definition of c_2 we conclude that $t_1 \leq t_0 + h \cdot c_2$. By adding this inequality and the inequality $t_0 + 2h \cdot (c_2 - c_1) < t_2$ in the hypothesis, we obtain the inequality $t_1 + h \cdot (c_2 - 2c_1) < t_2$. This is exactly the relationship between t_1 and t_2 which is required by Theorem 3.6 to ensure that T_2 returns a higher value than T_1. □

The next corollary also follows from Theorem 3.6:

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 returns a larger number than T_1.

Proof: If $c_2 \leq 2c_1$, then $h \cdot (c_2 - 2c_1) \leq 0$. By adding this inequality and the the inequality $t_1 < t_2$ we again obtain the relationship between t_1 and t_2 that allows us to use Theorem 3.6 to ensure that T_2 returns a higher value than T_1. □

Together with the definition of linearizability, this leads to our main *local* linerizability criteria for uniform networks:

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

This implies that Bitonic counting networks [4], Periodic counting networks [4], the networks of [18] and [9] are all linearizable for $c_2 \leq 2 \cdot c_1$. It also implies that counting and Diffracting trees [24] and the uniform trees of Busch and Mavronicolas [10] are linearizable for $c_2 \leq 2 \cdot c_1$.

We now consider a modification allowing to turn any uniform depth counting network into a linearizable network given that $c_2 \leq k \cdot c_1$ for some $k \geq 2$.

Corollary 3.10 Given a uniform counting network of depth h, another uniform counting network of depth $\lceil h \cdot (k-1) \rceil$ can be constructed so that it is linearizable for any $k \geq 2$ such that $c_2 \leq k \cdot c_1$.

Proof: Given the original network, we attach in front of each of its inputs a path of length $\lceil h \cdot (k-2) \rceil$ of 1-input 1-output "balancers" wired one after the other. The tokens traversing such balancers simply proceed from one to the next. For any two tokens that traverse the new network in a time-disjoint fashion, their

tbh

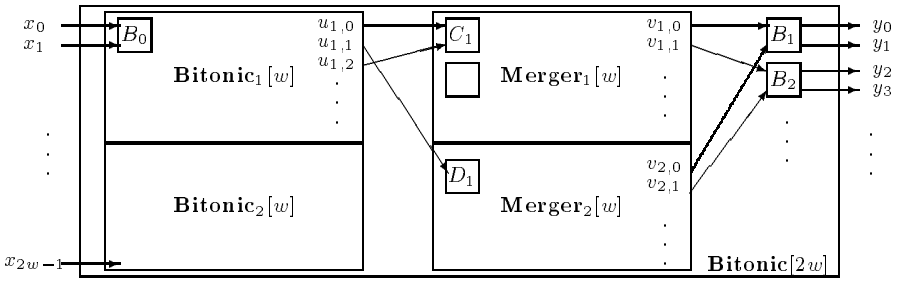

Fig. 4. Inductive step for Lemma 4.2.

traversals of the original (sub)network are such that the second token enters it at least $\lceil h(k-2)\rceil c_1 \geq h(c_2/c_1 - 2)c_1 = h \cdot c_2 - 2h \cdot c_1$ time after the first token exits. By Theorem 3.6, the second token returns a higher value. □

4. LIMITS ON LINEARIZABILITY OF TREES AND BITONIC COUNTERS

We now show some limitations on the linearizability of Diffracting trees [24] and Bitonic counting networks [4] by constructing execution scenarios under which they exhibit non-linearizable behavior.

Theorem 4.1 Counting and Diffracting trees are not linearizable if $c_2 > 2 \cdot c_1$.

Proof: Let h be the depth of the tree and let $\varepsilon > 0$ be such that $c_2 = (2+\varepsilon) \cdot c_1$. We consider an execution in which the first two tokens, T_0 and T_1, enter the tree at the same time t_0 (we visualize the tree on its side with its root to the left and the leaves on the right). Without loss of generality, let T_0 go up (corresponding to the root balancer transition from 0 to 1) and T_1 go down (the balancer transition from 1 back to 0), i.e., T_0 precedes T_1. After traversing the root, T_0 proceeds at the slowest possible pace of one wire per c_2 time, while T_1 proceeds at the fastest possible pace of one wire per c_1 time. T_1 reaches the topmost leaf of the bottom 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 1 wire per c_1 time. Of these tokens, 2^{h-1} tokens go to the upper subtree and the remaining $2^{h-1} - 1$ tokens go to the lower 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 of fast tokens reaches the leaves at time $t_3 = t_2 + h \cdot c_1 = t_1 + \delta + h \cdot c_1 = t_0 + 2h \cdot c_1 + \delta = t_0 + h \cdot (c_2 - c_1\varepsilon) + \delta = t_0 + h \cdot c_2 - c_1 h\varepsilon + \delta$. Since we chose δ such that $0 < \delta < \varepsilon$, the inequality can be further simplified to $t_3 < t_0 + h \cdot c_2 = t_4$. 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 top subtree, at least one token reaches the

topmost leaf of the tree and returns the value 0. This token traverses the counting tree completely after T_1 exits, but returns a smaller value. □

We now consider Bitonic networks.

Lemma 4.2 Let T_0 be the first token to enter a Bitonic counting network. Suppose T_0 enters through input X_0 and completely traverses the network alone. If subsequently tokens T_1 and T_2 enter the network in this order through X_0, then: (a) the balancer that is attached to X_0 is the only balancer that both T_1 and T_2 pass through, (b) T_0 exits through output wire Y_0, T_1 through output wire Y_1 and T_2 through output wire Y_2 (mod w).

Proof: By induction on the width w of the network: The 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 width $w \geq 2$, we prove that it holds for networks of width $2w$. The inductive step is depicted in Figure 4, and the balancer and exit labels below refer to that 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 an 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$[2w]$.

By the inductive hypothesis for Bitonic$_1[w]$, token 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 output $u_{1,0}$ is the same as $u_{1,2}$). By the construction of Bitonic$[2w]$, T_0 and T_2 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_2 via $v_{1,1}$, else Bitonic$[2w]$ will not reach a quiescent state in the execution where T_0 is the only token. Similarly, T_1 exits via $v_{2,0}$ of Merger$_2[w]$. In the final row of balancers, T_0 and T_1 traverse B_1, and T_2 traverses B_2.

To show (a) we observe that T_1 and T_2 may only traverse the same balancer inside Bitonic$_1[w]$, and by the inductive hypothesis, B_0 is the only such balancer.

To show (b), we observe that T_0 traverses the network alone and it reaches B_1 first and exits via Y_0, and so T_1 necessarily exits via Y_1. The only remaining token T_2 exits via Y_2. □

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

Proof: In the example in Section 1 we established that a network of width 2 consisting of a single balancer and two counters is not linearizable, and it is easy to see that this is so for any c_1 and c_2 such that $c_2 > 2 \cdot c_1$. Below we consider networks with $w > 2$. We choose $\varepsilon, \delta_1, \delta_2 > 0$ such that $\delta_1 + \delta_2 < \varepsilon$, and we let $c_2 = 2 \cdot c_1 + \varepsilon$.

Using the framework of Lemma 4.2, we deploy the three tokens T_0, T_1, and T_2 according to the following scenario. 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, at some time t_1, token T_1 also enters via X_0, 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 of 1 wire per c_2 time, while T_2 proceeds at the fastest possible pace of 1 wire per c_1 time. 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 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 these tokens' paths, they exit the network at time $t'_3 = t_3 + h \cdot c_1$. Since $\delta_1 + \delta_2 < \varepsilon$, these tokens exit before the slow token T_2.

During this execution, the network is traversed by $w + 3$ tokens. If no other tokens enter the network, then each of outputs Y_0, Y_1, and Y_2 has each two tokens that exit through it, and outputs $Y_3, \ldots, Y_{w-1}$ 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 results Section 5, when the ratio c_2/c_1 increases beyond 2, the percentage of non-linearizable operations also increases. Below we show that for Bitonic networks there can be a large fraction of tokens that exhibit non-linearizable behavior for certain ratios of 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. Moreover, for such c_2 and c_1 there exists an execution scenario with $3w/2$ tokens such that $w/2$ tokens result in non-linearizable operations.

Proof: The Bitonic counting network [4] of width w, Bitonic[w], has depth $h = \frac{\log w \cdot (\log w + 1)}{2}$. The network consists of two stages (see Figure 5). 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 the 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 [11]). Note that this inductive construction of the merger is different from, but isomorphic to the construction in Figure 4. The construction we use here yields a clearer proof.

A non-linearizable schedule is constructed as follows: The first wave of $w/2$ tokens enters Bitonic$_1$[$w/2$] network at the same time and proceeds 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 $\delta > 0$.

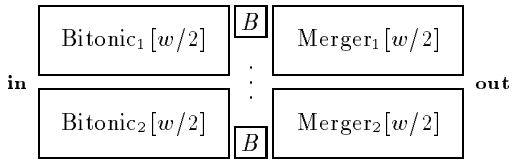

Fig. 5. Inductive construction of Bitonic$[w]$ for Theorem 4.4 (wires are omitted).

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

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

The third wave of $w/2$ tokens proceeds in lock step at the fastest pace of one wire per c_1 time to the exits. Therefore 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 wire out and returns counter values that are all lower than those obtained by the second wave. There are three waves of $w/2$ tokens out of which $w/2$ tokens are non-linearizable. □

We have shown specific scenarios in which the violations of local timing conditions lead to non-linearizable executions in important classes of uniform counting networks. The work of Mavronicolas et al. [20] shows how violations of timing conditions lead to non-linearizability in general counting networks (see Section 6).

5. EMPIRICAL EVALUATION OF LINEARIZABILITY

We evaluated the linearizability of counting networks on a simulated distributed-shared-memory machine similar to the MIT *Alewife* of Agarwal et al [1]. Alewife is a large-scale multiprocessor that supports cache-coherent distributed shared memory and user-level message-passing. The nodes communicate via messages on a two-dimensional mesh network. A Communication and Memory Management Unit on each node holds the cache tags and implements the memory coherence protocol by synthesizing messages to other nodes. Our experiments make use of the shared memory interface only. To simulate the Alewife we used *Proteus*[2], a multiprocessor simulator developed by Brewer, Dellarocas, Colbrook, and Weihl [8]. Proteus simulates parallel code by multiplexing several parallel threads on a single CPU.

[2]Version 3.00, dated February 18, 1993.

Each thread runs on its own virtual CPU with accompanying local memory, cache and communications hardware, keeping track of how much time is spent using each component. In order to facilitate fast simulations, Proteus does not do complete hardware simulations. Instead, operations which are local (do not interact with the parallel environment) are run uninterrupted on the simulating machine's CPU and memory. The amount of time used for local calculations is added to the time spent performing (simulated) globally visible operations to derive each thread's notion of the current time. Proteus makes sure a thread can only see global events within the scope of its local time.

Implementation and experimentation methodology

For our benchmarks, we implemented the Diffracting tree [24] and the Bitonic counting network [4] in shared memory. Both types of data structures gave each simulated processor with one of the two possible timing characteristics. The first kind allowed the processors to traverse the network unimpeded. The second kind introduced a time delay following the traversal of a balancer. This delay models the network delays or additional work that a processor may need to perform. We randomly designated a fraction of the processors, all of whom were be subjected to such delays. We performed two sets of experiments. In one set of experiments, the fraction F was 25%, in the other F was 50%. For each set of experiments, the time delay is defined via a *workload* variable W equal to $100, 1000, 10000$, and 100000 wait cycles .

We ran the scenarios varying the number of processors from 4, 16, 64, 128, 256, and up to 440 (this upper limit is due to the specifics of the hardware configuration we used). The execution of each simulation proceeded until each processor performed 200 operations. This number was chosen because of the long simulation times for large number of processors. (We also performed this test using 5,000 total operations). The graphs plot the non-linearizability ratio, i.e the percentage of non-linearizable operations (see Definition 2.6) among all the operations during the execution.

Every balancer was implemented as a critical section protected by a Mellor-Crummey and Scott (MCS) queue-lock [21] and, in the Diffracting tree, using a multi-prism implementation [23]. This was done to reduce contention on the balancers which would have attenuated the influence of the W-waiting periods on the c_2/c_1 relation.

The pseudocode for the main component of the simulation, the operation of obtaining the "next" counter value is given in Figure 6. This code was executed by each simulated process. `SharedCounter` is the concurrent counter implementation. In our simulations it was either the Bitonic counting network or the Diffracting tree counter implementation. The array `TotalIncrements` ensured that each processor performed `MaxIncrements` operations. The private variable, `old` and `new`, were used to respectively remember the previous value of the counter value obtained within the process, and to store the new value. All other variables are the global simulator variables. That means that all the processes could access them atomically

```
00  Reset(SharedCounter);
01  Array TotalIncrements[1..n] init {0:1..n};
02  GreatestNumber := -1;
03  for all processors id = 1 ... n cobegin
04     while TotalIncrements[id] <= MaxIncrements do
05        old := GreatestNumber;
06        new := Fetch&Increment(SharedCounter);
07        TotalIncrements[id] := TotalIncrement[id] + 1;
08        if new < old
09        then Nonlin := Nonlin + 1;
10        else GreatestNumber := max(new, GreatestNumber);
11        end if
12     end while
14  coend.
```

Fig. 6. Counter simulation main loop.

at no cost. `Nonlin` is the number of non-linearizable operations we observed.

A typical implementation of a shared-memory counter is shown in Figure 7.

```
    type balancer is
    begin
      state: regular or Diffracting balancer state
      next:   array [0..d-1] of ptr to balancer
    end

    constants
    width: global integer
    input : global ptr to some input wire of a Bitonic network or binary tree
             of balancers

 1  function fetch&incr(): integer
 2  begin
 3    b:= input
 4    while not leaf(b)
 5      b := traverse-balancer(b)
 6    endwhile
 7    i := increment_counter_at_leaf(b)
 8    return i * width + number_of_leaf(b)
 9  end
```

Fig. 7. A Shared-Memory tree-based counter implementation

We present the empirical data by charting the non-linearizability ratio as the function of the number of processors. In each of our experiments, we compute the average time it takes for a processor to traverse a balancer and a wire when the

workload $W = 0$. We use this average as the approximation of c_1 in the presentation. Note that using such average is conservative – for example, using the minimum value for such traversal would cause an increase in c_2/c_1 ratio and thus "excuse" or "explain" more of the non-linearizable operations observed in some scenarios. Using this definition of c_1, we compute c_2 as $(Average\text{-}c_1 + Workload)/Average\text{-}c_1 = 1 + Workload/Average\text{-}c_1$.

The absolute values of the average c_1 vary between the Bitonic network and the Diffracting tree due to the difference in the processing time associated with the prism in the Diffracting tree implementation. For ease of presentation, all data is normalized with respect to the average c_1 in the execution. To illustrate the ratio c_2/c_1 (c_1 divided by c_2) we present the normalized c_2 and also the normalized standard deviation for c_1 in the form *Standard-deviation / Average-*c_1.

Presentation and assessment of empirical data

The main results are presented in Figure 8 for the Diffracting tree and Figure 9 for the Bitonic network. The charts show the non-linearizability ratio as the function of the number of processors P. Each figure contains two charts, one showing the results with 25% delayed processors and the other with 50% delayed processors.

In Tables 1 and 2 we give the normalized c_2 for the Diffracting tree and the Bitonic network respectively. In Tables 3 and 4 we give the respective normalized standard deviations for c_1.

Using the theoretical results and empirical data we now discuss the effects of timing, network depth, concurrency, and asynchrony and randomization on the linearizability of the simulated execution scenarios.

The effects of timing. As can be seen, for the lower delay workloads ($W = 100$ and $W = 1000$), the normalized c_2 is less then or close to 2, and no linearizability violations occur for 16 or more processors. For these workloads some non-linearizability is observed for small number of processors, i.e., four. Note that for the Bitonic network, the violations occur for these values of W when the normalized c_2 is above 5. Even so, the non-linearizability ratio here is less than 1%.

For higher delay workloads ($W = 10000$ and $W = 100000$), the normalized c_2 is well above 2 and for the Bitonic network it reaches several hundreds (see Tables 1 and 2). As expected, we observe significant increase in the ratio of non-linearizable operations. For the Diffracting tree the ratios peak at about 26% for 16 processors 50% of which incur delays of $W = 100000$. For the Bitonic network the peak ratio is about 12% for the same parameters. Substantially lower peak non-linearizable ratios, of 10% and 5% respectively, are observed for $F = 25\%$ and 16 processors.

It is surprising is that despite the high c_2, the non-linearizable token ratio falls sharply as the number of processors is increased. We examine some of the reasons for this phenomena.

Diffracting Tree – Normalized c_2												
P :	440	256	128	64	16	4	440	256	128	64	16	4
W	25% of processes delayed						50% of processes delayed					
100	1.10	1.10	1.09	1.08	1.08	1.08	1.10	1.10	1.09	1.08	1.08	1.08
1000	2.02	2.01	1.88	1.77	1.77	1.73	2.04	2.00	1.86	1.75	1.76	1.72
10000	11.28	10.78	9.43	8.43	8.76	8.35	11.16	10.16	8.81	8.09	8.49	8.46
100000	105.54	98.72	84.48	74.61	78.30	74.34	103.51	90.12	76.44	70.14	76.38	81.14

Table 1. Normalized c_2 in the simulations of Diffracting trees.

Bitonic Network – Normalized c_2												
P :	440	256	128	64	16	4	440	256	128	64	16	4
W	25% of processes delayed						50% of processes delayed					
100	1.10	1.14	1.18	1.22	1.34	1.40	1.10	1.14	1.19	1.22	1.34	1.40
1000	2.02	2.38	2.83	3.19	4.44	5.02	2.08	2.43	2.89	3.31	4.51	5.13
10000	12.37	16.29	19.70	22.99	35.31	41.42	13.90	17.53	20.69	24.16	36.15	43.49
100000	120.78	159.63	191.96	224.29	345.52	405.26	148.00	179.39	205.60	240.85	357.78	431.28

Table 2. Normalized c_2 in the simulations of Bitonic networks.

Diffracting Tree – Normalized Standard Deviation												
P :	440	256	128	64	16	4	440	256	128	64	16	4
W	25% of processes delayed						50% of processes delayed					
100	0.65	0.55	0.60	0.63	0.69	0.70	0.73	0.56	0.61	0.63	0.69	0.71
1000	0.59	0.56	0.61	0.64	0.70	0.72	0.58	0.57	0.62	0.65	0.70	0.72
10000	0.59	0.57	0.61	0.64	0.69	0.71	0.58	0.60	0.62	0.64	0.70	0.71
100000	0.56	0.57	0.61	0.64	0.69	0.72	0.57	0.59	0.62	0.64	0.70	0.69

Table 3. Standard deviation normalized for average c_1 for the simulations of Diffracting trees

Bitonic Network – Normalized Standard Deviation												
P :	440	256	128	64	16	4	440	256	128	64	16	4
W	25% of processes delayed						50% of processes delayed					
100	1.86	0.48	0.47	0.47	0.33	0.33	0.43	0.48	0.46	0.46	0.32	0.33
1000	1.93	0.51	0.49	0.46	0.30	0.34	0.48	0.52	0.49	0.44	0.27	0.33
10000	2.00	0.51	0.47	0.46	0.30	0.35	0.65	0.56	0.48	0.44	0.28	0.34
100000	2.29	0.49	0.47	0.45	0.29	0.34	0.58	0.51	0.46	0.43	0.27	0.34

Table 4. Standard deviation normalized for average c_1 for the simulations of Bitonic networks.

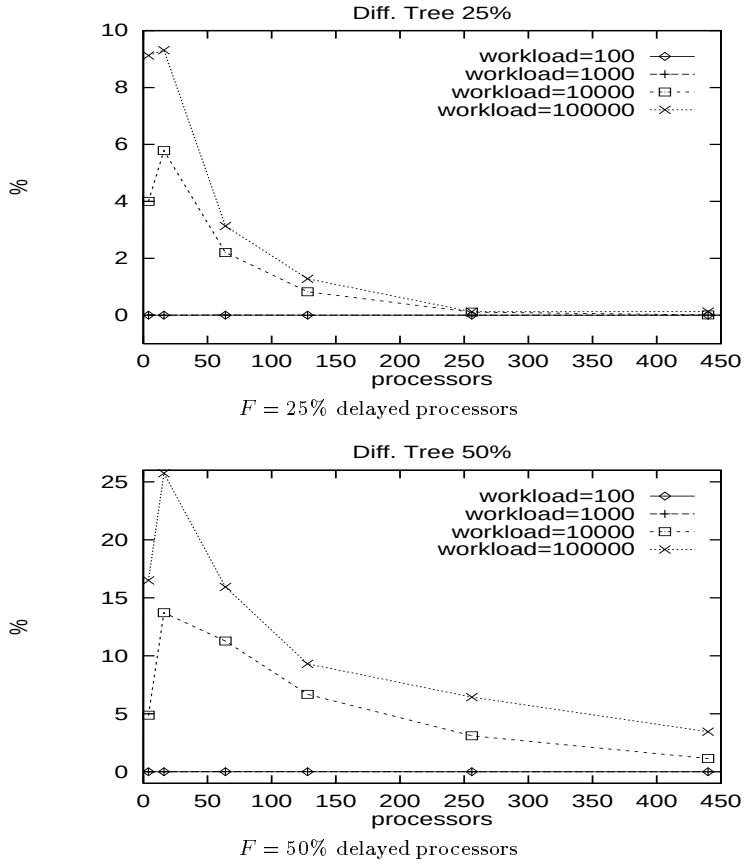

Fig. 8. Non-linearizability Ratios for the Diffracting Tree.

The effects of network depth. The Bitonic networks have substantially greater depth than Diffracting trees of the same width. This results in many more operations overlapping in the Bitonic networks given identical token arrival schedules. With this differentiating factor, we expect and indeed observe substantially fewer linearizability violations in the Bitonic network simulations as compared to the Diffracting tree simulations. This padding effect is also suggested by Theorem 3.6 that enables, for a known $c_2/c_1 > 2$, the construction of a linearizable networks by extending the depth of any known counting network.

The effects of concurrency. There are simple scenarios that, using as few processors as 2, produce high levels of non-linearizability. Recall our example in Section 1, in which three tokens caused one non-linearizable operation. Let processor P_0 be the owner of the token T_0 and processor P_1 be the owner of tokens T_1 and T_2. If the token T_0 is very slow, so that it does not exit the network for a long time, then any sequence of tokens T_i generated by P_0 will have each of its even-numbered tokens T_{2j} return lower counter values than its odd-numbered tokens T_{2j-1} for $j > 0$. This is because the even- and odd-numbered tokens traverse the network sequentially. If

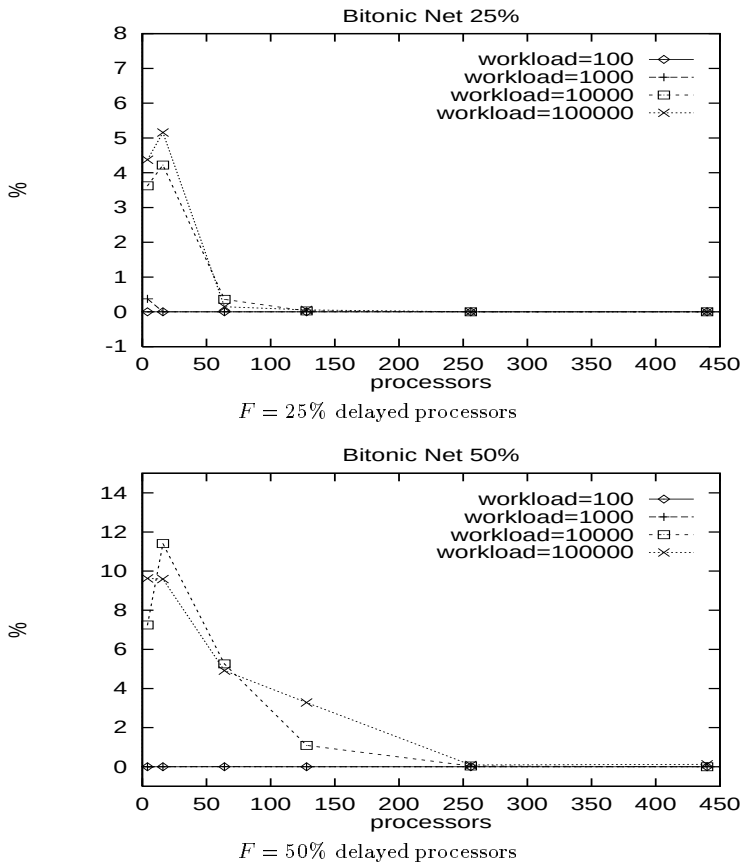

Fig. 9. Non-linearizability Ratios for the Bitonic Network.

there were three processors, such that T_{2j} is concurrent with T_{2j-1}, then the there would be no nonlinearizable operations.

Although far from a complete characterization, the above observation of linearizability versus concurrency provides intuition for why there is a dramatic reduction, at high concurrency, in the number of non-linearizable operations for both the Diffracting tree and the Bitonic network.

Of course the counting network approach is optimized for high concurrency, so it is not surprising that deploying counting networks in low-concurrency setting has its drawbacks. For few processors, there are more efficient and linearizable solutions [14].

The effects of asynchrony and randomization. We also tested the linearizability of our implementation when either all or no tokens were delayed, i.e., the cases of $F = 0\%$ and $F = 100\%$, and/or when the additional delays were eliminated, i.e., $W = 0$. In none of these simulation were there any non-linearizable operations. Although not surprising – these scenarios create timing schedules close to those

of an implementation that is synchronous – we performed these simulations for completeness.

In another simulation scenario we forced every token to wait a random number of cycles between 0 and W. Again, the simulation was observed to be completely linearizable. Randomization apparently has attenuating effect that prevents consistent accumulation of timing discrepancies by faster or slower tokens.

6. CONCLUSIONS AND DISCUSSION

Our paper studies the effects of timing on the linearizability of *uniform* counting networks. Our results were recently extended and generalized by Mavronicolas, Papatriantafilou, and Tsigas [20], to include non-uniform networks. For a given network G, let d be the maximum path length from inputs to outputs, and s be the shortest such path. They show that a counting network is linearizable if $c_2/c_1 \leq 2s/d$ (for uniform networks $s = d$, and the linearizability requirement reduces to the $c_2/c_1 \leq 2$ shown in Section 3). Furthermore, they introduce the powerful notion of an *influence radius* of a graph G, $irad_G$, as the length of the maximum common subpath of any two maximal paths from an internal balancer to any two outputs, and show that a network is not linearizable if $c_2/c_1 > d/irad_G + 1$ (for uniform networks $irad_G = d$, and linearizability is violated when $c_2/c_1 > 2$ as we show here).

We have considered *local* timing characteristics at balancers. The linearizablility question can also be posed in terms of *global* timing characteristics, i.e., in terms of the minimum and maximum time it takes a token to traverse the entire network and without the restriction on the time to traverse each individual balancer. Our examination of Counting trees and Bitonic networks shows that violations of required local conditions lead to non-linearizable executions (this is also shown for general networks in [20]). In these executions we use tokens that traverse a network at the fastest and the slowest possible paces. The fast tokens "bypass" the slow tokens only at the exits. Therefore even if the required conditions are *global*, our scenarios still yield non-linearizable executions.

There are many other variations of the timing model which one may investigate. However, we feel the most interesting direction to follow at this time is the characterization of applications that do not have an absolute requirement for linearizability, that is, ones requiring that only a given fraction of the operations be linearizable.

7. ACKNOWLEDGMENTS

The authors thank Maurice Herlihy for insightful comments and the anonymous referees for helpful suggestions.

REFERENCES

[1] Anant Agarwal, Ricardo Bianchini, David Chaiken, David Kranz Kirk Johnson, John Kubiatowicz, Beng-Hong Lim, Ken Mackenzie, and Donald Yeung. The MIT Alewife machine:

Architecture and performance. In *Twenty-Second International Symposium on Computer Architecture*, pages 2–13, Santa Margherita, Ligure, Italy, June 1995.

[2] E. Aharonson and H. Attiya. Counting networks with arbitrary fan out. Distributed Computing, 8(4):163–169, 1995. Also: Technical Report 679, The Technion, June 1991. Earlier version in [2].

[3] B. Aiello, R. Venkatesan, and M. Yung. Coins, weights and contention in balancing networks. In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 193–214, Los Angeles, CA, August 1994.

[4] James Aspnes, Maurice Herlihy, and Nir Shavit. Counting networks. *Journal of the ACM*, 41(5):1020–1048, September 1994. Earlier version in *Proceedings of the 23rd ACM Annual Symposium on Theory of Computing*, pp. 348–358, May 1991. Also, MIT Technical Report MIT/LCS/TM-451, June 1991.

[5] Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *Journal of the ACM*, 41(1):122–152, January 1994.

[6] Hagit Attiya, Nancy Lynch, and Nir Shavit. Are wait-free algorithms fast? *Journal of the ACM*, 41(4):725–763, July 1994.

[7] K. E. Batcher. Sorting networks and their applications. *Proceedings of AFIPS Spring Joint Computer Conference*, pages 307–314, 1968.

[8] E. A. Brewer, C. N. Dellarocas, A. Colbrook, and W. E. Weihl. Proteus: a high-performance parallel-architecture simulator. Technical Report MIT/LCS/TR-516, MIT Laboratory for Computer Science, September 1991.

[9] C. Busch and M. Mavronicolas. A combinatorial treatment of balancing networks. In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 206–215, Los Angeles, CA, August 1994.

[10] C. Busch and M. Mavronicolas. New bounds on depth and contention for counting networks, October 1995. Preprint, Univ. of Cyprus.

[11] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, Cambridge, MA/New York, 1990.

[12] E.W. Felten, A. LaMarca, and R. Ladner. Building counting networks from larger balancers. Technical Report TR-93-04-09, University of Washington, April 1993.

[13] James R. Goodman, Mary K. Vernon, and Philip J. Woest. Efficent synchronization primitives for large-scale cache-coherent multiprocessors. In *Proceedings of the 3rd International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 64–75, Boston, Massachusetts, April 1989.

[14] M. Herlihy, B. H. Lim, and N. Shavit. Scalable concurrent counting. *ACM Transactions on Computer Systems*, 13(4):343–364, 1995. Earlier version in *Proceedings of the Third Annual ASM Symposium on Parallel Algorithms and Architectures (SPAA)*, San Diego, CA, pages 219-227, July 1992. Full version available as DEC TR.

[15] Maurice Herlihy, Nir Shavit, and Orli Waarts. Low contention linearizable counting. In *Proceedings of the 32nd Annual Symposium on the Foundations of Computer Science (FOCS)*, pages 526–535, San Juan, Puerto Rico, October 1991. IEEE. Detailed version with empirical results appeared as MIT Technical Memo MIT/LCS/TM-459, November 1991.

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

[17] M. Klugerman. *Small-Depth Counting Networks*. PhD thesis, MIT, Cambridge, MA 02139, 1994.

[18] M. Klugerman and C. G. Plaxton. Small-depth counting networks. In *Proceedings of the 24th ACM Symposium on Theory of Computing (STOC)*, pages 417–428, 1992.

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

[20] M. Mavronicolas, M. Papatriantafilou, and P. Tsigas. The impact of timing on linearizability in counting networks, 1997. To appear in the 11th International Parallel Processing Symposium, Geneva, Switzerland.

[21] J. M. Mellor-Crummey and M. L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 9(1):21–65, February 1991. Earlier version published as TR 342, University of Rochester, Computer Science Department, April 1990, and COMP TR90-114, Center for Research on Parallel Computation, Rice UNIV, May 1990.

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

[23] Nir Shavit and Dan Touitou. Elimination trees and the construction of pools and stacks. In *SPAA'95: 7th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 54–63, Santa Barbara, California, July 1995. Also, Tel-Aviv University *Technical Report*, January 1995.

[24] Nir Shavit and Asaph Zemach. Diffracting trees. *ACM Transactions on Computer Systems*, 14(4):385–428, November 1996.

[25] Paul M.B. Vitanyi. Distributed elections in an archimedean ring of processors. In *Proc. of the 16th ACM Symposium on Theory of Computing*, pages 542–547, 1984.