Sequentially Consistent versus Linearizable Counting Networks

(Extended Abstract)

Marios Mavronicolas[*] *Michael Merritt*[†] *Gadi Taubenfeld*[‡]

Abstract

We compare the impact of timing conditions on implementing *sequentially consistent* and *linearizable* counters using *counting networks* in distributed systems. For counting problems in application domains which do not require linearizability but will run correctly if only sequential consistency is provided, the potential payoffs of our investigation are threefold: First, we show that sequential consistency and linearizability cannot be distinguished by the timing conditions previously considered in the context of counting networks, and thus in contexts in which these constraints apply, it is possible to rely on the stronger semantics of linearizability, which simplifies proofs and enhances compositionality. Second, we identify local timing conditions that support sequential consistency but not linearizability, and thus suggest weaker, easily implementable timing conditions that are likely to be sufficient in many applications. Third, we show that any kind of synchronization that is too weak to support even sequential consistency, may violate it significantly for some counting networks; hence, we identify timing conditions that are to be totally ruled out for specific applications that rely critically on either sequential consistency or linearizability.

[*]Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269-3155. Part of the work of this author was performed at Department of Computer Science, University of Cyprus (supported by funds for the promotion of research at University of Cyprus), and while at AT&T Labs – Research, as a visitor to the Special Year on Networks, DIMACS Center for Discrete Mathematics and Theoretical Computer Science, Piscataway, NJ. Email: mavronic@engr.uconn.edu

[†]AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932-0971. Email: mischu@research.att.com

[‡]The Open University, 16 Klausner St., Tel-Aviv 61392, Israel. Part of the work was performed while visiting AT&T Labs – Research. Email: gadi@cs.openu.ac.il

1 Introduction

1.1 Overview

The *counting problem* is to design a protocol in which a number of concurrent processes repeatedly acquire successive values. An additional possible requirement, called *linearizability*, imposes that the order of the assigned values reflects the real-time order in which they were requested [HW90]. Linearizable counting can be used as a building block in basic constructions such as concurrent time-stamps generation, implementing FIFO buffers, and efficient shared counters.

Counting networks are highly concurrent data structures which solve the (non-linearizable) counting problem, in a way that reduce sequential bottlenecks and contention [AHS94]. They are implemented in shared memory as networks of records (balancers) and pointers (wires). It is known that there does not exist a completely asynchronous counting network, with finite depth, that guarantees linearizability in *all* possible schedules [HSW96]. Thus, previous work has addressed the question of identifying appropriate timing conditions that outlaw non-linearizable schedules, thus rendering a counting network linearizable [HSW96, LSST96, MPT97].

Sequential consistency is a consistency condition weaker than linearizability [L79]. For counting networks, it assures that for any two tokens (i.e., requests for values) by the same process, the earlier token obtains a smaller value than the later one. This natural monotonicity property is reasonable to expect from a counter primitive. Moreover, distinguishing sequential consistency from linearizability requires interprocess communication outside of the shared counter primitive. Although the standard correctness condition for shared memory multiprocessors, the important notion of sequential consistency has not been investigated previously in the context of counting networks. As we will see later, there are counting networks which, under specific timing conditions, satisfy sequential consistency but not linearizability.

1.2 Summary of results

In this work, we demonstrate that previously studied timing conditions fail to distinguish sequential consistency from linearizability. We introduce a new local timing condition and demonstrate that it suffices to guarantee sequential consistency, but not linearizability. Finally, we show that previous measures of the "fraction" of inconsistent counter operations can be applied to sequential consistency. We show that in the worst case, weak timing assumptions previously shown to admit proportion of incorrect (non-linearizable) operations [LSST96], actually admit the same proportion of non-sequentially consistent operations–a large proportion of locally-observable inconsistencies.

For counting problems that originate from application domains which do not absolutely require linearizability but which run correctly if only sequential consistency is provided, the potential payoffs of our investigation are threefold: First, if it turns out that both sequential consistency and linearizability are supported by the timing constraints that are inherent in any particular distributed environment, one can rely instead on the stronger semantics of linearizability, since this simplifies proofs and enhances compositionality. Second, an understanding of weak timing conditions that support sequential consistency but not linearizability allows the designer to make available the timing condition that is "cheapest," yet sufficient for each specific application. Third, an understanding of the negative effect of timing conditions that are too "weak" to support sequential consistency (and hence, also violate linearizability), can help the designer of a distributed system to choose the most "cost-effective" timing condition for those applications that are even willing to occasionally sacrifice sequential consistency in order to achieve improved performance.

The results presented in this work are:

- We discuss several timing conditions (regulating the rate at which processes move through a counting network, and global inter-operation delays) and show that considering only these conditions *cannot* distinguish linearizability from sequential consistency (Theorem 3.2). Previous work on timing conditions for assuring linearizability in counting networks involve only these timing conditions [HSW96, LSST96, MPT97]. Thus, it follows that previously known results (especially necessary conditions) hold also for sequential consistency.

- We identify timing conditions (involving an additional bound on *local* inter-operation delay) that *can* distinguish linearizability from sequential consistency. That is, we present timing conditions that are sufficient for sequential consistency but not for linearizability (Theorem 4.1). Thus, for example, for any given uniform counting network, we present timing conditions under which the network is sequentially consistent but is not linearizable. Moreover, these local timing conditions are straightforward to implement.

- The *fraction* of non-sequentially-consistent (resp., non-linearizable) operations in a finite execution is defined to be the minimum number of operations whose removal yields a sequentially consistent (resp., linearizable) execution, divided by the number of completed operations in the execution. We present results, both upper and lower bounds, on these fractions (Section 5).

1.3 Related work

Counting networks, with balancers with fan-in and fan-out two, were first introduced and investigated in [AHS94]. A generalization was introduced in [AA95] where topological constraints on the design using larger balancers were investigated. Similar design issues were investigated in [FLL93] and [HKM93]. The notion of *linearizability* is presented in [HW90]. Linearizable counting networks assure that the order of values returned by the same process or by different processes reflects the real-time order in which they were requested. The fact that there does not exist a completely asynchronous counting network, with finite depth, that guarantees linearizability in all possible schedules is pointed out in [HSW96]. Various results on concurrent counting (without using counting networks) are reported in [BMT95, MT97, MTY96].

The first work to investigate timing constraints on the behavior of counting networks and to identify an appropriate timing condition which guarantees linearizability is [LSST96]. Moreover, this work shows that this sufficient condition is also a necessary condition for bitonic networks and counting trees. Additional results in this direction are presented in [MPT97]. We discuss specific results from these papers in Section 2.3.

The notion of *sequential consistency* is introduced in [L79]. Sequentially consistent counting networks assure that the order of values returned by the same process reflects the real-time order in which they were requested. We know of no previous work on sequentially consistent counting networks. The impact of timing conditions on the relative costs of implementing linearizability and sequential consistency in message-passing has been investigated in [AW94, MR92].

2 Preliminaries

We first give the definitions of balancing and counting networks, then we define several timing parameters. We conclude with definitions of consistency conditions and review previous results.

2.1 Balancing and counting networks

Many of the definitions in this section are adapted from [AA95, AHS94, LSST96, MPT97]. Balancing networks are constructed from elements, called balancers, that direct *tokens* from inputs to outputs, and wires, which acyclically interconnect the balancers [AHS94]. An (f_{in}, f_{out})*-balancer,* or *balancer* for short, is a routing element receiving tokens on f_{in} input wires, and sending out tokens to f_{out} output wires; the integers f_{in} and f_{out} are called the balancer's *fan-in* and *fan-out,* respectively. Processes introduce tokens on the balancer's input wires at arbitrary times, after some delay shepherd them instantaneously through the balancer, arriving on an output wire. Roughly speaking, a balancer acts as a round-robin scheduler, taking a stream of input tokens and forwarding them to its output wires, from top to bottom; thus, a balancer effectively "balances" input tokens on its output wires. The wires are interconnection and delay elements, but provide no queueing or ordering of pending tokens.

For each index i, $0 \leq i \leq f_{in}$, we denote by x_i the *history variable* that stands for the number of tokens that have entered on input wire i; for each index j, $0 \leq j \leq f_{out}$, we similarly denote by y_j the *history variable* that stands for the number of tokens that have exited on output wire j. In the *initial state,* all wires are empty. A balancer is *quiescent* if $\sum_{i=1}^{f_{in}} x_i = \sum_{j=1}^{f_{out}} y_j$; that is, in a quiescent state, the number of tokens that exited the balancer is equal to the number of tokens that entered it. The following are important properties of an (f_{in}, f_{out})-balancer:

1. *Safety property:* $\sum_{i=1}^{f_{in}} x_i \geq \sum_{j=1}^{f_{out}} y_j$; that is, a balancer never creates tokens spontaneously.

2. *Liveness property:* If only a any finite number of tokens are input to the balancer, then eventually the balancer reaches a quiescent state, $\sum_{i=1}^{f_{in}} x_i = \sum_{i=1}^{f_{out}} x_i$; that is, a balancer never "swallows" tokens.

3. *Step property:* For any pair of indices j and k such that $1 \leq j < k \leq f_{out}$, $0 \leq y_j - y_k \leq 1$.

A (w_{in}, w_{out})*-balancing network* is a directed, acyclic graph G with three kinds of nodes: (1) w_{in} *source* nodes $X_1, X_2, \ldots, X_{w_{in}}$, (2) w_{out} *sink* nodes $Y_1, Y_2, \ldots, Y_{w_{out}}$, and (3) a finite number of *inner* nodes. The source and sink nodes represent the input and output wires of the network, respectively, while the inner nodes represent the balancers of the network. The edges of G connect the balancers by identifying the input and output wires of successive balancers; thus, a node that corresponds to an (f_{in}, f_{out})-balancer has f_{in} incoming edges and f_{out} outgoing edges, that coincide with the input and output wires of the balancer. Moreover, the outgoing and incoming degrees of all source and sink nodes, respectively, are equal to one, while the incoming and outgoing degrees of all source and sink nodes, respectively, are equal to zero.

The *size* of a balancing network is the total number of its inner nodes. For any wire z in a balancing network, the *depth* of z, denoted $d(z)$, is defined to be zero if z is an input wire connected to a source node, and $\max_{1 \leq l \leq f_{in}} d(z_l)+1$, for an output wire of an (f_{in}, f_{out})-balancer with input wires $z_1, z_2, \ldots, z_{f_{in}}$. For any balancer b in a balancing network, the *depth of* b, denoted $d(b)$, is the maximal wire depth, over all of its output wires. A *layer* in a balancing network is a maximal set of balancers that have the same depth. The *depth* of a balancing network G, denoted $d(G)$ or d for short, is the maximum balancer depth, over all of its balancers. For any integer l, $1 \leq l \leq d(G) + 1$, the l-th layer of G is the collection of nodes (balancers or sinks) whose depth is l. A *path* in a balancing network G is defined in the natural way. A balancing network is *uniform* [LSST96, Definition 2.1] if each node of the network lies on some path from source nodes and sink nodes, and all paths from source nodes to sink nodes have the same length.

The safety and liveness properties for a balancing network follow naturally from those for its balancers. Thus, if only finitely many tokens enter a balancing network, it eventually reaches a *quiescent state* in which all tokens that entered the network have exited (reached a sink). Since processes shepherd tokens through different parts of the network at different times, the step property is only required of such quiescent states. However, not all balancing networks satisfy the step property. A (w_{in}, w_{out})*-counting network* is a (w_{in}, w_{out})-balancing network for which, in any quiescent state, for any pair of indices j and k such that $1 \leq j < k \leq w_{out}$, $0 \leq y_j - y_k \leq 1$; that is, in quiescent states the output of a counting network has the *step property.* Each one of the w_{out} sink nodes of a counting network is identified with an atomic *counter.* The tokens exiting on output wire y_j, $1 \leq j \leq w_{out}$, are consecutively assigned by the counter residing there the integers j, $j + w_{out}$, $j + 2w_{out}$, and so on. We remark that known constructions of counting networks [AA95, AVY94, AHS94, BHM94, BM98, FLL93, HKM93, KP92] are uniform.

On a multiprocessor shared memory machine, a balancing network is implemented as a shared memory data structure, where balancers are records and wires are pointers from one record to another. Each process runs a program that repeatedly performs an increment operation on the network by traversing the data structure from some input pointer to some output pointer, each time shepherding a new *token* through the network. We assume an unbounded set of processes assigned to each input wires: all tokens generated by a specific process enter on the assigned input wire. A process shepherds a token through the network by atomically updating each balancer, and using the returned value to choose which pointer to follow. For simplicity, we assume that balancer updates are instantaneous, and all delays occur on the wires.

2.2 Timing conditions

An *execution* of a balancing network G can be denoted as a (possibly infinite) sequence $\mathcal{E} = e_1, e_2, \ldots$ of instantaneous transition events e_i of the form $BAL_p(T_j, b_k, f_m, f_n)$ (corresponding to a token T_j of process p traversing a balancer b_k, entering on input wire f_m and exiting on output wire f_n) or $COUNT_p(T_j, C_k, i)$ (corresponding to a token T_j of process p traversing a counter C_k, obtaining the value i). A *timed execution $R_\mathcal{E}$ for an execution $\mathcal{E}$ of a balancing network G* associates a time with each event in the execution, in a non-decreasing sequence, denoted $R_\mathcal{E} = \langle e_1, t_1 \rangle, \langle e_2, t_2 \rangle, \ldots$ Moreover, if the execution $\mathcal{E}$ is infinite, then the sequence $t_1, t_2, \ldots$ is unbounded.

Let $T(\mathcal{E})$ be the set of tokens appearing in execution $\mathcal{E}$. A timed execution $R_\mathcal{E}$ determines a *schedule* $S_\mathcal{E} : T(\mathcal{E}) \times [d(G)+1] \rightarrow \Re$ that specifies for any token $T \in T(\mathcal{E})$ and layer l, $1 \leq l \leq d(G)+1$, the time $S_\mathcal{E}(T, l)$ at which token T passes through a node in layer l.
Associated with a schedule $S_\mathcal{E}$ of a network G are the following *timing parameters*:

$\mathbf{c^P_{min}}$ – *lower bound on wire delay for process P.* The minimum over all tokens T inserted by process P and all layers l, of the difference between the time at which T passes through layer l, and the time at which T passes through layer $l-1$, where $1 < l \leq d(G)+1$. Intuitively, c^P_{min} represents the minimum delay a token by process P "experiences" over any individual wire.

$\mathbf{c_{min}}$ – *lower bound on wire delay.* The minimum over all processes P of c^P_{min}. Intuitively, c_{min} represents the minimum delay a token "experiences" over any individual wire.

$\mathbf{c_{max}}$ – *upper bound on wire delay.* The maximum over all tokens T and all layers l, of the difference between the time at which T passes through layer l, and the time at which T passes through layer $l-1$, where $1 < l \leq d(G)+1$. Intuitively, c_{max} represents the maximum delay a token "experiences" over any individual wire.

$\mathbf{C^P_L}$ – *lower bound on local inter-operation delay for process P.* The minimum over all pairs of consecutive tokens T and T' by process P, of the difference between the time at which token T' passes through layer 1 of the network, and the time at which token T passes through layer $d(G)+1$. Intuitively, C^P_L measure the "local delay" incurred between the time a token by P exits the network and the time a new token by P can enter it.

$\mathbf{C_L}$ – *lower bound on local inter-operation delay.* The minimum, over all processes P, of C^P_L. Intuitively, C_L measure the "local delay" incurred between the time some token exits the network and the time a new token by the same process can enter it.

$\mathbf{C_g}$ – *lower bound on global delay.* The minimum over all pairs of tokens T and T' that do not *overlap* (are not inside the network at the same time) of the difference between the time at which token T' passes through layer 1 of the network, and the time at which token T passes through layer $d(G)+1$. Intuitively, C_g measures the "global delay" incurred between the time some token exits the network and the time a new token (possibly by another process) can enter it.

The timing parameters c_{min}, c_{max}, and C_g were introduced and their relationship to linearizability was studied by Lynch *et al.* [LSST96]. The timing parameters c^P_{min}, C^P_L, and C_L were previously considered in work by Shavit *et al.* studying the impact of local delay on global performance, but not in the context of assuring consistency conditions [SUZ98].

2.3 Consistency conditions

A *serialization of execution $\mathcal{E}$* is a total order of the tokens in $T(\mathcal{E})$ that respects the order of tokens at each individual process. A timed execution $R_\mathcal{E}$ specifies a partial order $\xrightarrow{R_\mathcal{E}}$ on tokens in $T(\mathcal{E})$ as follows: For any tokens T and T' in $T(\mathcal{E})$, $T \xrightarrow{R_\mathcal{E}} T'$ if and only if T completely precedes T' in the execution $R_\mathcal{E}$.

Herlihy *et al.* [HSW96] adapted the definition of linearizability from [HW90] to balancing networks: A *linearization* of timed execution $R_\mathcal{E}$ is a serialization of $\mathcal{E}$ that extends $\xrightarrow{R_\mathcal{E}}$. That is, for any tokens T and T' in $\mathcal{E}$, if $T \xrightarrow{R(\mathcal{E})} T'$, then T precedes T' in the linearization. A timed execution $R_\mathcal{E}$ is *linearizable* if it admits a linearization in which every token receives a value greater than that of all tokens earlier in the linearization. A balancing network is *linearizable* [HSW96] under a timing condition if every timed execution satisfying that condition is linearizable.

Lynch *et al.* showed in [LSST96], among other results, that a uniform counting network is linearizable if for any two tokens traversing the network, their traversals either overlap or they are separated by time

$t > d(G)(c_{max} - 2c_{min})$. Thus, if $d(G)(c_{max} - 2c_{min}) < C_g$ (and hence also if $c_{max}/c_{min} \leq 2$) then such a network is linearizable. As the authors point out, the bound $d(G)(c_{max} - 2c_{min}) < C_g$ is not a local condition– it would require coordination among individual processes to ensure the C_g bound is preserved. Hence, the stronger bound $c_{max}/c_{min} \leq 2$ is stressed as a local linearizability criteria.

In Section 4 we show that weaker, local timing bounds suffice to guarantee the weaker correctness condition of sequential consistency, which we adapt from [L79] to balancing networks. Say that a timed execution $R_{\mathcal{E}}$ of a balancing network G is *sequentially consistent* if the successive token traversals by each process return increasing values. A balancing network is *sequentially consistent* under a timing condition if every timed execution satisfying that condition is sequentially consistent.

For any execution $\mathcal{E}$, consider the *restriction* of $\mathcal{E}$ to events of process P, denoted $\mathcal{E} \mid P$. Clearly, this restriction inherits the order of tokens at process P (already determined by execution $\mathcal{E}$). Say that an execution $\mathcal{E}$ is *sequentially consistent with respect to process* P if the values obtained by tokens in the restriction $\mathcal{E} \mid P$ are in increasing order. A balancing network G is *sequentially consistent with respect to process* P if every execution of it is is *sequentially consistent with respect to process* P.

Proposition 2.1 *Assume that for each process P, the balancing network G is sequentially consistent with respect to process P. Then, G is sequentially consistent.*

3 Timing conditions which do not distinguish linearizability and sequential consistency

In this section, we demonstrate that limiting c_{min}, c_{max} and C_g *cannot* distinguish linearizability from sequential consistency.

The proof of this result depends on the modular counting carried out by individual fan-out-f balancers; that is, f tokens can be simultaneously carried through a balancer without affecting the other tokens. The lemma below formalizes this property.

Lemma 3.1 *Let $R_{\mathcal{E}} = \langle e_1, t_1\rangle, ..., \langle e_i, t_i\rangle, ...$ be a timed execution of a balancer B with fan-out f. Suppose:*

- *$p_1, ..., p_f$ are processes shepherding tokens $T_1, ..., T_w$, respectively,*
- *$y = j \pmod f$ tokens have passed through B after step $\langle e_i, t_i\rangle$ of $R_{\mathcal{E}}$,*
- *$k_1, ..., k_f$ are indices of input wires of B, and*
- *$t_i \leq t$ and either $\langle e_i, t_i\rangle$ is the last step of $R_{\mathcal{E}}$ or $t \leq t_{i+1}$, and (in either case) let β be the suffix of $R_{\mathcal{E}}$ starting after step $\langle e_i, t_i\rangle$.*

Then $\langle e_1, t_1\rangle, ..., \langle e_i, t_i\rangle$ $\langle BAL_{p_1}(T_1, B, k_1, j+1), t\rangle, ...,$ $\langle BAL_{p_{f-j}}(T_{f-j}, B, k_{f-j}, f), t\rangle$ $\langle BAL_{p_{f-j+1}}(T_{f-j+1}, B, k_{f-j+1}, 1), t\rangle, ...,$ $\langle BAL_{p_f}(T_j, B, k_f, j), t\rangle, \beta$ is a timed execution of B.

Proof: A balancer with fan-out f acts as a counter modulo f. Since exactly w new tokens pass through B simultaneously, later tokens are unaffected. ■

Theorem 3.2 *A uniform counting network under timing conditions c_{min}, c_{max} and C_g is linearizable if and only if it is sequentially consistent.*

Proof: Since linearizability implies sequential consistency, it suffices to show that for any uniform counting network G, if there is a timed execution $R_{\mathcal{E}}$ of G that is not linearizable and satisfies a timing condition c_{min}, c_{max} and C_g, then there is a timed execution $R'_{\mathcal{E}}$ that is not sequentially consistent and yet satisfies the same timing condition.

Let $R_{\mathcal{E}}$ be such a non-linearizable timed execution. For simplicity, assume for now that each balancer has the same number of input and output wires. The proof below constructs the timed execution $R'_{\mathcal{E}}$ from $R_{\mathcal{E}}$. Since $R_{\mathcal{E}}$ is not linearizable, it must contain two serial operations for tokens T_1 and T_2, such that the first token, T_1, receives some value y_{big} while the token T_2 following it receives a smaller value, y_{small}. If these two tokens belong to the same processor, then $R_{\mathcal{E}}$ is already not sequentially consistent.[1]

So assume otherwise. The proof demonstrates that by carefully introducing and scheduling additional tokens, using the modular properties of balancers noted in Lemma 3.1, a timed execution $R'_{\mathcal{E}}$ can be constructed in which two tokens associated with the same process mimic the behavior of T_1 and T_2, emerging with the values y_{big} and y_{small}, respectively.

Let W be the width of G, let $p_1, ..., p_W$ be processors that take no steps in $R_{\mathcal{E}}$, each p_i assigned to the input wire w_i, and suppose that the processor p associated with token T_1 is assigned the j'th input wire of G. Let $R_{\mathcal{E}_1}$ be the sequence obtained by relabelling the steps of token T_1 with processor index p_j. It should be clear that $R_{\mathcal{E}_1}$ is a timed execution of G with the same timing properties as $R_{\mathcal{E}}$: we have simply replaced a token of processor p with one by p_j.

Let D be the depth of D, let q be the processor that moves token T_2 through the network, and

[1] The original definition of counting networks [AHS94] allows each process to introduce tokens to an input wire that is either preassigned or chosen arbitrarily. In the second case, the claim follows trivially by relabelling tokens T_1 and T_2 with a process that otherwise takes no steps in $R_{\mathcal{E}}$.

let $R_{\mathcal{E}_1} = \langle e_1, t_1\rangle$, ..., $\langle BAL_q(T_2, B_1, in_1, out_1), t_{q_1}\rangle$, ..., $\langle BAL_q(T_2, B_{D-1}, in_{D-1} in_{D-1}, out_{D-1}), t_{q_{D-1}}\rangle$, ..., $\langle COUNT_q(T_2, C_D, y_{small}), t_{q_D}\rangle$, ..., where the identified events are the D steps q takes to move the token T_2 through the network.

Let $R_{\mathcal{E}_2}$ be the prefix of $R_{\mathcal{E}_1}$ that ends just before event $\langle COUNT_q(T_2, C_D, y_{small}), t_{q_D}\rangle$.

Observe that in any counting network there must be a path from every input wire to every output wire. (To see this, note that the counting properties must hold even if every token comes in over a single input wire.) In particular, there is a path from input wire j to the counter C_D. We can use Lemma 3.1 to route a token by p_j along this path, emerging just before T_2, and returning the value y. To prevent this token from affecting others, we move W tokens synchronously through the network, moving through each layer of the uniform network at the same speed as T_2. Specifically, just before $\langle BAL_q(T_2, B_1, in_1, out_1), t_{q_1}\rangle$ in the execution, and with the same time t_{q-1}, we add W events, one for each p_i, routing p_j through the first balancer on the path to C_D. The result is a timed execution $R_{\mathcal{E}_3}$ that is an execution of each component balancer and counter, and so of G. Since a token was moved on every input wire through its first balancer, there is now a token on every output wire of the first layer of the network, and hence a token on every input wire of the next layer. So again before event $\langle BAL_q(T_2, B_2, in_2, out_2), t_{q_2}\rangle$ we can add W events, one for each p_i, routing p_j through the second balancer on the path to C_D, resulting in a sequence $R_{\mathcal{E}_4}$ that is an execution of each component balancer and counter, and so of G. Continuing, for $D-1$ steps, we end with a sequence $R_{\mathcal{E}_{D+1}}$ that is a timed execution of each component balancer and counter, and so of G, in which p_j's token $T_{p_j}$is on the input wire to counter C_D. Finally, timed execution $R_{\mathcal{E}_{D+2}}$ is produced by appending $\langle COUNT_{p_j}(T_{p_j} C_D, y_{small}), t_{q_D}\rangle$ to $R_{\mathcal{E}_{D+1}}$. Moreover, since each of the new tokens move through the network at exactly the same rate as T_2, $R_{\mathcal{E}_{D+2}}$ satisfies the same timing constraints as $R_{\mathcal{E}}$. But processor p_j performed two serial operations that returned y_{big} and then y_{small}, so $R_{\mathcal{E}_{D+2}}$ is not sequentially consistent.

Up to this point, the argument has assumed that each balancer has the same number of input and output wires. If this is not the case, then a similar construction will work, but many more than W tokens may be needed.

Let LCM_i be the least common multiple of the fan-out of the balancers in layer i of G. Focusing on the first layer of the network, if we put LCM_1 tokens on each input wire and route them as before simultaneously through the first layer, then at least one token will emerge on each output wire. As important, the number of tokens moving through each balancer will be a multiple of the fan-out of that balancer, as Lemma 3.1 requires.

To get at least one token on each output of the second layer, it suffices to put $LCM_2 LCM_1$ tokens on each input wire to G, and once again the number of tokens moving through each balancer is a multiple of the fan-out of that balancer. Finally, $\prod_{i=1}^{D-1} LCM_i$ tokens on each input wire of G will suffice to route the specific token for p_j to C_D.

Although there are (far) more tokens on each wire than in the argument above, they all move at the same rate as T_1. Hence the resulting execution satisfies the same constraints c_{min}, c_{max} and C_g. ■

The results reported in [HSW96, LSST96] identified timing conditions dependent only on the parameters c_{min}, c_{max}, and C_g, that are either necessary or sufficient (or both) for linearizability. Theorem 3.2 allows for the extension of such results to sequential consistency. For example, the following corollaries follow from Theorem 3.2 and similar results proved for linearizability in [LSST96, MPT97].

Corollary 3.3 *A bitonic counting network is sequentially consistent under timing conditions c_{max} and c_{min} if and only if $c_{max} \leq 2c_{min}$.*

If G is a uniform counting network, then we denote by $irad(G)$ the maximum, over every pair of output wires j and k of G, of the distance from j to the least common ancestor of j and k in G.

Corollary 3.4 *A uniform counting network G is sequentially consistent under timing conditions c_{max} and c_{min} only if $c_{max}/c_{min} \leq d(G)/irad(G) + 1$.*

Notice that the local delay C_L, is not explicitly mentioned in any statement of this section. However, for an arbitrary uniform counting network G, Corollary 3.4 implies that for some small enough local delay (say 0), G is not sequentially consistent; in the next section, we prove a theorem (Theorem 4.1) which implies that for some big enough local delay, G is sequentially consistent.

4 Timing conditions which distinguish linearizability and sequential consistency

In this section, we demonstrate that any uniform counting network G is sequentially consistent under the timing condition $d(G) \cdot (c_{max} - 2c_{min}) < C_L$, but that this condition is insufficient to imply linearizability. Unlike the global delay bound $d(G) \cdot (c_{max} - 2c_{min}) < C_g$, (which implies linearizability [LSST96]) this condition can be implemented easily using local clocks.

Theorem 4.1 *Let G be a uniform counting network and let c_{min}, c_{max} and C_L be timing conditions such that $d(G) \cdot (c_{max} - 2c_{min}) < C_L$. Then G is sequentially consistent under these conditions.*

To prove the theorem, we use the following result due to Lynch *et al.* [LSST96].

Proposition 4.2 ([LSST96]) *Assume that tokens T and T' traverse a uniform counting network G during the intervals $[t_{in}, t_{out}]$ and $[t'_{in}, t'_{out}]$, respectively. If $d(G) \cdot (c_{max} - 2c_{min}) < t'_{in} - t_{out}$, then T' returns a higher value than T.*

Proposition 4.2 can be immediately extended as follows:

Corollary 4.3 *Assume that tokens T and T', both of process P, traverse a uniform counting network G during the intervals $[t_{in}, t_{out}]$ and $[t'_{in}, t'_{out}]$, respectively. If $d(G) \cdot (c_{max} - 2c^P_{min}) < t'_{in} - t_{out}$, then T' returns a higher value than T.*

Lemma 4.4 *Consider any uniform counting network G, and any process P.*
If $d(G) \cdot (c_{max} - 2c^P_{min}) < C^P_L$, then G is sequentially consistent for process P.

Proof: Consider any tokens T and T', both of process P, traversing G during the intervals $[t_{in}, t_{out}]$ and $[t'_{in}, t'_{out}]$, with T preceding T'. By assumption,

$$d(G) \cdot (c_{max} - 2c^P_{min}) < C^P_L. \tag{1}$$

By definition of C^P_L,

$$C^P_L \leq t'_{in} - t_{out}. \tag{2}$$

From (1) and (2) it follows that,

$$d(G) \cdot (c_{max} - 2c^P_{min}) < t'_{in} - t_{out}. \tag{3}$$

It follows from (3) and Corollary 4.3 that T' returns a higher value than T. Since T and T' were chosen arbitrarily, this implies that G is sequentially consistent for process P. ∎

The proof of Theorem 4.1 follows from Lemma 4.4 and Proposition 2.1. It follows from Theorem 4.1 and Corollary 3.4, that there are timing conditions for any uniform counting network that imply sequential consistency but not linearizability:

Corollary 4.5 *For any uniform counting network, G, there are timing conditions c_{min}, c_{max} and C_L such that under these constraints G satisfies sequential consistency but does not satisfy linearizability.*

Proof: Let G be any uniform counting network and c_{min} and c_{max} timing conditions such that $c_{max}/c_{min} > d(G)/irad(G) + 1$. By Corollary 3.4, there exists a timed execution $R_{\mathcal{E}}$ of G satisfying these timing conditions that is not sequentially consistent. Now rename the processes that shepherd more than one token in $\mathcal{E}$ in such a way that each token is shepherded by a different process, resulting in a timed execution $R'_{\mathcal{E}}$. Since $R_{\mathcal{E}}$ is not sequentially consistent, the construction implies that $R'_{\mathcal{E}}$ is not linearizable. Now let C_L be any value such that $C_L > d(G)(c_{max} - 2c_{min})$. By construction, $R'_{\mathcal{E}}$ vacuously satisfies $d(G)(c_{max} - 2c_{min}) < C_L$, as needed to complete the proof. ∎

5 Inconsistency fractions

5.1 Definitions

Say that a token T is *non-linearizable in an execution* $\mathcal{E}$ [LSST96], if there exists some other token T', which completely precedes T and returns a value higher than that of T. Say that a token T is *non-sequentially consistent in an execution* $\mathcal{E}$ if there exists some other token T', shepherded by the same process, which precedes T and returns a value higher than that of T.

For any finite execution $\mathcal{E}$ of a balancing network G, the *non-linearizability fraction of* $\mathcal{E}$ [LSST96] is defined to be the number of non-linearizable tokens in $\mathcal{E}$ divided by the total number of tokens in $T(\mathcal{E})$. The *non-linearizability fraction of* G (under a given set of timing conditions) denoted $\mathbf{F}_{nl}(G)$ is the maximum, over all executions $\mathcal{E}$ of G satisfying the timing conditions, of the non-linearizability fraction of $\mathcal{E}$.

Similarly, the *non-sequentially consistency fraction of* $\mathcal{E}$ is the number of non-sequentially consistent tokens in $\mathcal{E}$ divided by the total number of tokens in $T(\mathcal{E})$. The *non-sequential consistency fraction of* G, (under a given set of timing conditions) denoted $\mathbf{F}_{nsc}(G)$, is the maximum, over all executions $\mathcal{E}$ of G satisfying the timing conditions, of the non-sequential consistency fraction of $\mathcal{E}$. Clearly, $\mathbf{F}_{nl}(G) \geq \mathbf{F}_{nsc}(G)$.

The *absolute non-linearizability fraction of* $\mathcal{E}$ is defined to be the number of non-linearizable tokens in $\mathcal{E}$ whose removal yields a linearizable execution, divided by the total number of tokens in $\mathcal{E}$. The *absolute non-linearizability fraction of* G, (under a given set of timing conditions) denoted $\mathbf{AF}_{nl}(G)$, is the maximum, over all executions $\mathcal{E}$ of G satisfying the timing conditions, of the absolute non-linearizability fraction of $\mathcal{E}$. Clearly, $\mathbf{F}_{nl}(G) \geq \mathbf{AF}_{nl}(G)$.

Similarly, the *absolute non-sequential consistency fraction of* $\mathcal{E}$ is the number of non-sequentially consistent tokens in $\mathcal{E}$ whose removal yields a sequentially consistent execution divided by the total number of tokens in $\mathcal{E}$. The *absolute non-sequential consistency frac-*

tion of G, denoted $\mathbf{AF}_{nsc}(G)$, is the maximum, over all executions $\mathcal{E}$ of G, (under a given set of timing conditions) of the absolute non-sequential consistency fraction of $\mathcal{E}$ satisfying the timing conditions. Clearly, $\mathbf{F}_{nsc}(G) \geq \mathbf{AF}_{nsc}(G)$, and $\mathbf{AF}_{nl}(G) \geq \mathbf{AF}_{nsc}(G)$. There is only a single known lower bound on $\mathbf{F}_{nl}(G)$ for the particular case where G is the bitonic counting network [AHS94], and under a particular timing assumption that involves the size of the network. The following results is due to Lynch *et al.* [LSST96]:

Proposition 5.1 ([LSST96]) *Let $G^{(w)}$ be the bitonic network with width w, and let c_{min} and c_{max} be timing conditions such that $c_{max}/c_{min} > (3+\lg w)/2$. Then under these conditions $\mathbf{F}_{nl}(G^{(w)}) \geq 1/3$.*

We can extend Proposition 5.1, using a similar construction, as follows:

Proposition 5.2 *Let $G^{(w)}$ be the bitonic network with width w, and let c_{min} and c_{max} be timing conditions such that $c_{max}/c_{min} > (3+\lg w)/2$. Then under these conditions $\mathbf{F}_{nsc}(G^{(w)}) \geq 1/3$.*

5.2 An upper bound

The next theorem is an upper bound on the absolute non-sequential consistency fraction, under a timing assumption expressing "bounded asynchrony".

Theorem 5.3 *Let G be a uniform counting network, ℓ an integer greater than 1, and c_{min} and c_{max} timing conditions such that $c_{max}/c_{min} \leq \ell$. Then, under these conditions,*

$$\mathbf{AF}_{nsc}(G) \leq \frac{\ell-2}{\ell-1}.$$

We first show a technical claim.

Lemma 5.4 *Let G be a uniform counting network, ℓ an integer greater than 1, and c_{min} and c_{max} timing conditions such that $c_{max}/c^P_{min} \leq \ell$. Let $T_1, ..., T_\ell$ be a sequence of tokens of P such that T_i starts before T_{i+1} for all $1 \leq i < \ell$. Then, T_1 obtains a smaller value than T_ℓ.*

Proof: For each token, it takes at least $d(G)\cdot c^P_{min}$ time units to go through the network. Thus, since there are $\ell-2$ tokens between T_1 and T_ℓ, there is a local delay of $C > (\ell-2)\cdot d(G)\cdot c^P_{min}$ between the time at which T_1 exits the network and the time at which T_ℓ enters it. By Corollary 4.3, T_ℓ returns a higher value than T_1, if $d(G)\cdot(c_{max} - 2c^P_{min}) < C$. Thus, T_ℓ returns a higher value than T_1, if $d(G)\cdot(c_{max} - 2c^P_{min}) \leq (\ell-2)\cdot d(G)\cdot c^P_{min}$. Since, by assumption, $c_{max}/c^P_{min} \leq \ell$, this inequality always holds. ■

We continue with the proof of the Theorem 5.3.

Proof: For any execution $\mathcal{E}$ of G and process P, let $\mathcal{E}^P$ denote the sequence of tokens shepherded by P in $\mathcal{E}$. By Lemma 5.4, for any two tokens T_i and T_j such that T_i appears ℓ positions before T_j in $\mathcal{E}^P$, T_i obtains a smaller value than T_j. Thus, if for any process P, we remove from $\mathcal{E}$ each token its position modulo $(\ell-1)$ (in $\mathcal{E}^P$) is different from 1, we get a sequentially consistent timed execution. ■

5.3 A lower bound

Next we presnt a lower bound on the non-sequential consistency fraction of any counting network that has a certain topological property. For lack of space, all the proofs are omitted. We first give some appropriate definitions.

For any balancer output wire j in a network G, define the *valency of j in G,* denoted $\mathsf{Valency}(j)$, to be the set of sink nodes reachable from j. For any balancer b in a network G, the *valency of b in G,* denoted $\mathsf{Valency}(b)$, is the union of the valencies of its output wires. Clearly, for any counting network, for any particular layer of it, every sink node must be reachable from some node in the layer. Hence, we have:

Proposition 5.5 *Fix any layer ℓ in a counting network G with fan-out w_{out}, where $1 \leq \ell \leq d(G)$. Then, $\bigcup_{b\in\ell} \mathsf{Valency}(b) = \{1, 2, \ldots, w_{out}\}$.*

Consider any balancer b in a network G, with output wires $1, 2, \ldots, f_{out}$. Say that *b is univalent in G* if for each pair of indices j and k, $1 \leq j, k \leq f_{out}$, $\mathsf{Valency}(j) \bigcap \mathsf{Valency}(k) = \emptyset$. Intuitively, b is univalent if each of its output wires unambiguously determines a set of possible output wires of the network, those that can be reached by a token starting from that particular output wire of b. Say that *a layer ℓ is univalent in G* if each of the balancers in ℓ is univalent in G.

For any pair of sets of integers V_1 and V_2, say that V_1 *precedes* V_2, denoted $\mathsf{V}_1 \prec \mathsf{V}_2$, if every integer in V_1 is less than any integer in V_2. Say that *b induces a total precedence in G* if the set of sets $\mathsf{Valency}(1), \mathsf{Valency}(2), \ldots, \mathsf{Valency}(f_{out})$ is totally ordered with respect to $\prec$; that is, for each pair of indices j and k, $1 \leq j, k \leq f_{out}$, either $\mathsf{Valency}(y_j) \prec \mathsf{Valency}(y_k)$ or $\mathsf{Valency}(y_k) \prec \mathsf{Valency}(y_j)$. Clearly, any balancer that induces a total precedence is also univalent, but not vice versa. Intuitively, b induces a total precedence if it leads to eventual "decisions" that do not "cross" each other. Say that *a layer ℓ induces a total precedence in G* if each of the balancers in ℓ induces a total precedence in G.

For any network G, define the *splitting depth of G,* denoted $sd(G)$, to be either the minimum integer ℓ, $1 \leq \ell \leq d(G)$, such that layer ℓ of G induces a total precedence in G, or infinite if no such integer exists; intuitively, the split of G measures "how far" into G a token needs to get before the set of possible output wires it will exit from is unambiguously determined. A *splittable network* is a network with finite splitting depth. Consider any splittable network G. Say that G is *uniformly splittable* if for each balancer b in layer $sd(G)$, for any output wires j and k of b, $|\mathsf{Valency}(j)| = |\mathsf{Valency}(k)|$. Clearly, for any uniformly splittable counting network $G^{(w_{out})}$ made up of balancers of fan-in and fan-out two, for any balancer b in layer $sd(G)$ with output wires 1 and 2, $|\mathsf{Valency}(1)| = \{1, 2, \ldots, w_{out}/2\}$ and $|\mathsf{Valency}(2)| = \{w_{out}/2+1, w_{out}/2+2, \ldots, w_{out}\}$. We continue to demonstrate that the original counting networks introduced in [AHS94], namely the bitonic and periodic counting networks, are uniformly splittable; moreover, we shall calculate their splitting depths. We start by showing:

Proposition 5.6 *Let $B^{(w)}$ be the bitonic counting network of width w. Then, $B^{(w)}$ is uniformly splittable and $sd(B^{(w)}) = (\lg^2 w - \lg w + 2)/2$.*

We also show corresponding results for the periodic counting network.

Proposition 5.7 *Let $P^{(w)}$ be the periodic counting network of width w. Then, $P^{(w)}$ is uniformly splittable and $sd(P^{(w)}) = \lg^2 w - \lg w + 1$.*

For any network G such that $sd(G) < d(G)$, define the *split suffix of* G, denoted $\mathsf{Ssuffix}(G)$, to be the suffix of G consisting of layers $sd(G)+1, sd(G)+2, \ldots, d(G)$ of G.

half

For any uniformly splittable network G made up of balancers of fan-in and fan-out two, we provide an inductive construction of a finite sequence of networks $\mathsf{Split}^{(0)}(G), \mathsf{Split}^{(1)}(G), \ldots$, which we call the *splitting sequence for* G, as follows. For the basis case, $\mathsf{Split}^{(0)}(G) = G$. Assume inductively that we have defined the network $\mathsf{Split}^{(\ell-1)}(G)$ for some integer $\ell \geq 1$. We proceed to the induction step. If $sd(\mathsf{Split}^{(\ell-1)}(G)) \geq d(\mathsf{Split}^{(\ell-1)}(G))$, then the construction terminates; else, define $\mathsf{Split}^{(\ell)}(G)$ to be the network $\mathsf{Ssuffix}(\mathsf{Split}^{(\ell-1)}(G))_b$. Intuitively, the construction starts with the network G, and each network subsequently in the sequence results by "chopping off" the preceding network at its splitting depth, if that is possible, and taking the bottom part of the "chopped" split suffix. Since G is uniformly splittable, $sd(G) \leq d(G)$; thus, it follows that the length of the sequence $\mathsf{Split}^{(0)}(G), \mathsf{Split}^{(1)}(G), \ldots$, is no less than two and at most $d(G)$. Say that G is *uniformly splittable all the way through* if each network but the last in the splitting sequence for G is uniformly splittable.

Assume that the splitting sequence for G has length greater than one. Define the *splitting number of* G, denoted *split*(G), to be the length of the splitting sequence for G. We proceed to calculate the splitting numbers of the original bitonic and periodic counting networks [AHS94]. We start by showing:

Proposition 5.8 *Let $B^{(w)}$ be the bitonic counting network of width w. Then, $B^{(w)}$ is uniformly splittable all the way through, and $split(B^{(w)}) = \lg w - 1$.*

We continue to show:

Proposition 5.9 *Let $P^{(w)}$ be the periodic counting network of width w. Then, $P^{(w)}$ is uniformly splittable all the way through, and $split(P^{(w)}) = \lg w - 1$.*

We proceed to show our main lower bound based on the splitting number.

Theorem 5.10 *Consider any uniform counting network $G^{(w)}$ that is uniformly splittable all the way through. Then, for any integer ℓ, $1 \leq \ell \leq split(G^{(w)})$, and timing conditions c_{min} and c_{max}, if*

$$\frac{c_{max}}{c_{min}} \;>\; 1 + \frac{d(G^{(w)})}{d(\mathsf{Split}^{(\ell)}(G^{(w)}))}\,,$$

then,

$$\mathbf{F}_{nl}(G^{(w)}) \;\geq\; \frac{2^\ell - 1}{2 \cdot 2^\ell - 1}\,,$$

and

$$\mathbf{F}_{nsc}(G^{(w)}) \;\geq\; \frac{1}{2 \cdot 2^\ell - 1}\,.$$

We remark that Theorem 5.10 establishes a collection of lower bounds on the non-linearizability and non-sequential consistency fractions, one for each possible value of ℓ and under a different timing assumption, in the form of a lower bound, on c_{min} and c_{max}; this assumption depends on ℓ since it involves $d(\mathsf{Split}^{(\ell)}(G^{(w)}))$. As ℓ increases, $d(\mathsf{Split}^{(\ell)}(G^{(w)}))$ decreases, and the assumed lower bound on c_{max}/c_{min} therefore increases.

By Propositions 5.6, 5.7, 5.8, and 5.9, Theorem 5.10 immediately implies the following for the case where ℓ is taken to be equal to the splitting number of the network.

Corollary 5.11 *Consider the bitonic counting network $B^{(w)}$. Assume that*

$$\frac{c_{max}}{c_{min}} \;>\; 1 + \frac{\lg w(\lg w + 1)}{2}\,.$$

Then,

$$\mathbf{F}_{nl}(B^{(w)}) \;\geq\; \frac{w-2}{2(w-1)}\,,$$

and

$$\mathbf{F}_{nsc}(B^{(w)}) \quad \geq \quad \frac{1}{w-1} \, .$$

Corollary 5.12 *Consider the periodic counting network $P^{(w)}$. Assume that*

$$\frac{c_{max}}{c_{min}} \quad > \quad 1 + \lg^2 w \, .$$

Then,

$$\mathbf{F}_{nl}(P^{(w)}) \quad \geq \quad \frac{w-2}{2(w-1)} \, ,$$

and

$$\mathbf{F}_{nsc}(P^{(w)}) \quad \geq \quad \frac{1}{w-1} \, .$$

We remark that the lower bounds on the non-linearizability fractions established in Corollaries 5.11 and 5.12 tend to $1/2$ (from below) as w tends to infinity, while the corresponding lower bounds for sequential consistency tend to 0. This suggests to use large counting networks for applications that are willing to occasionally sacrifice sequential consistency, in case it is expensive to provide a timing constraint that would guarantee sequential consistency in all schedules.

References

[AA95] E. Aharonson and H. Attiya, "Counting networks with arbitrary fan-out," *Distributed Computing,* Vol. 8, pp. 163–169, 1995.

[AVY94] W. Aiello, R. Venkatesan and M. Yung, "Coins, weights and contention in balancing networks," *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing,* pp. 193–205, August 1994.

[AHS94] J. Aspnes, M. Herlihy and N. Shavit, "Counting networks," *Journal of the ACM,* Vol. 41, No. 5, pp. 1020–1048, September 1994.

[AW94] H. Attiya and J. L. Welch, "Sequential consistency versus linearizability," *ACM Transactions on Computer Systems,* Vol. 12, No. 2, pp. 91–122, May 1994.

[BMT95] H. Brit, S. Moran and G. Taubenfeld, "Public data structures: counters as a special case," *Proceedings of the Third Israel Symposium on Theory of Computing and Systems,* Tel Aviv, January 1995.

[BHM94] C. Busch, N. Hardavellas and M. Mavronicolas, "Contention in counting networks," *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing,* p. 404, August 1994.

[BM98] C. Busch and M. Mavronicolas, "An Efficient Counting Network," *Proceedings of the 1st Merged International Parallel processing Symposium and IEEE Symposium on Parallel and Distributed Processing,* pp. 380–385, May 1998.

[FLL93] E. W. Felten, A. LaMarca and R. Ladner, "Building counting networks from larger balancers," Technical Report TR-93-04-09, Department of Computer Science and Engineering, University of Washington, April 1993.

[HKM93] N. Hardavellas, D. Karakos and M. Mavronicolas, "Notes on sorting and counting networks," *Proceedings of the 7th International Workshop on Distributed Algorithms (WDAG-93),* LNCS Vol. 725, Springer-Verlag, pp. 234–248, September 1993.

[HSW96] M. Herlihy, N. Shavit and O. Waarts, "Linearizable counting networks," *Distributed Computing,* Vol. 9, pp. 193–203, 1996.

[HW90] M. Herlihy and J. Wing, "Linearizability: A correctness condition for concurrent objects," *ACM Transactions on Programming Languages and Systems,* Vol. 12, No. 3, pp. 463–492, July 1990.

[KP92] M. Klugerman and C. G. Plaxton, "Small-Depth Counting Networks," *Proceedings of the 24th Annual ACM Symposium on Theory of Computing,* pp. 417–428, May 1992.

[L79] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs," *IEEE Transactions on Computers,* Vol. C-28, No. 9, pp. 690–691, September 1979.

[LSST96] N. Lynch, N. Shavit, A. Shvartsman and D. Touitou, "Counting networks are practically linearizable," *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing,* pp. 280–289, May 1996.

[MPT97] M. Mavronicolas, M. Papatriantafilou and Ph. Tsigas, "The impact of timing on linearizability in counting networks," *Proceedings of the 11th International Parallel Processing Symposium,* pp. 684–688, April 1997.

[MR92] M. Mavronicolas and D. Roth, "Efficient, Strongly Consistent Implementations of Shared Memory," *Proceedings of the 6th International Workshop on Distributed Algorithms (WDAG'92),* pp. 346–361, LNCS Vol. 486, Springer-Verlag, November 1992.

[MT97] S. Moran and G. Taubenfeld, "A lower bound on wait-free counting," *Journal of Algorithms,* Vol. 24, pp. 1–19, 1997.

[MTY96] S. Moran, G. Taubenfeld and I. Yadin, "Concurrent counting," *Journal of Computer and System Sciences,* Vol. 53, No. 1, pp. 61–78, August 1996.

[SUZ98] N. Shavit, E. Upfal and A. Zemach, "A steady state analysis of diffracting trees," *Theory of Computing Systems,* Vol. 31, No. 4, pp. 403–423, July/August 1998.