Sequentially Consistent versus Linearizable Counting Networks*

Marios Mavronicolas† *Michael Merritt*‡ *Gadi Taubenfeld*§

(August 13, 2007)

*A preliminary version of this work appears in the *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing (PODC 1999),* pp. 133–142, May 1999. This work has been partially supported by the IST Program of the European Union under projects DELIS (contract number 001907) and AEOLUS (contract number 15964).

†Department of Computer Science, University of Cyprus, Nicosia CY-1678, Cyprus. Currently visiting Faculty of Computer Science, Electrical Engineering and Mathematics, University of Paderborn, Paderborn, Germany. Part of the work of this author was performed at the Department of Computer Science and Engineering, University of Connecticut, Storrs, CT, and while at AT&T Labs – Research, Florham Park, NJ, as a visitor to Special Year on Networks, DIMACS Center for Discrete Mathematics and Theoretical Computer Science, Piscataway, NJ. Email: mavronic@ucy.ac.cy

‡AT&T Labs – Research, 180 Park Ave., Florham Park, NJ 07932-0971. Email: mischu@research.att.com

§The Interdisciplinary Center, P. O. Box 167, Kanfai Nesharim St., Herzliya 46150, Israel. Part of the work was performed while visiting AT&T Labs – Research, Florham Park, NJ. Email: tgadi@idc.ac.il

Abstract

We compare the impact of timing conditions on implementing *sequentially consistent* and *linearizable* counters using *(uniform) 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 results of our investigation, and their potential payoffs, 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; thus, in contexts where 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; thus, we 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.

Keywords: Counting networks, balancing networks, sequential consistency, linearizability, inconsistency fractions.

1 Introduction

1.1 Motivation and Overview

A solution to the *counting problem* is a protocol which allows a number of concurrent *processes* to repeatedly acquire successive *values* from a given range, such as addresses of memory locations or destination ids in a routing network. A standard consistency condition, *linearizability,* requires that the order of the values returned to (possibly different) processes reflect the real-time order of the request operations [HW90]. That is, a request operation that was completed earlier must obtain a smaller value than one invoked after its completion. Linearizable counting can be used as a building block in basic constructions such as barrier synchronization* [MS91], concurrent queues and stacks [FG91] and efficient shared program counters [ML89].

Counting networks are highly concurrent data structures which solve a (non-linearizable) counting problem in a way that reduces sequential bottlenecks and contention [AHS94]. Unlike *queue-locks* [MS91] and *combining trees* [GVW89], which were based on handing out values from a single memory location, counting networks hand out values from a collection of locations. To guarantee that the returned values are correct (that is, they exhibit no duplications or gaps), counting networks use a special network structure that is traversed by processes before they reach the counters; this structure is made up of *balancers* and *wires,* implemented in shared memory as records and pointers, respectively, whose role is to coordinate the routing of processes through it. The obvious advantage of counting networks is that they achieve spreading of the processes through a network structure, thereby reducing contention and increasing concurrency.

It is known that there does not exist a completely asynchronous counting network (with finite depth) that guarantees linearizability in *all* possible schedules [HSW96, Theorem 5.1]. So, it is natural to seek appropriate timing conditions that outlaw non-linearizable schedules, thus rendering a counting network linearizable [LSST99, MPT97].

Sequential consistency [L79] is a consistency condition weaker than linearizability. For counting networks, it assures that the order of values returned to the *same* process reflects the real-time order in which the values were requested. This natural monotonicity property is reasonable to expect from a counter primitive. However, although a standard consistency condition for shared memory multiprocessors, sequential consistency has not previously been investigated in the context of counting networks. Our work is a first step in such an investigation.†

*A *barrier synchronization* is a coordination mechanism that forces processes participating in a concurrent algorithm to wait until each one of them has reached a certain *coordination point* in its program. Using barriers often enables to significantly simplify the design of concurrent algorithms.

†However, we point out that a trivial modification to the proof for [HSW96, Theorem 5.1] immediately

In a nutshell, we demonstrate that previously studied timing conditions fail to distinguish sequential consistency from linearizability in the context of counting networks. Furthermore, we introduce a new *local* timing condition and establish 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 as well.

We remark that there are counting problems arising from practical applications that can do with only sequentially consistent counters. Consider, for example, a particular counter-based implementation of barrier synchronization for n concurrent processes, which uses a single counter C (initially set to 0). As soon as a process reaches the barrier, it increments the counter by 1 and "busy-waits"; when a process reads that $\mathsf{C} = n$, it signals to the other processes that they may run past the barrier (cf. [T06, Section 5.2]. A linearizable counter suffices; then, the last process to increment will get the value n and signal to the others. However, a sequentially consistent counter will also suffice; in that case, exactly one process will (still) get the value n (once all processes have started their increments) and all processes will run past the barrier.

1.2 Contribution

For counting problems that originate from application domains which do not absolutely require linearizability but will run correctly if only sequential consistency is provided, the results of our investigation, and their potential payoffs, 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; thus, in contexts where 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; thus, we 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.

yields that there does not exist a completely asynchronous counting network (with finite depth) that guarantees sequential consistency in all possible schedules.

[‡]Linearizability is a *compositional* condition: a system of objects is linearizable if and only if each individual object is linearizable [HW90]. Sequential consistency is not a compositional condition.

The specific results of our investigation contribute directly to these intended payoffs as follows:

- We consider several timing conditions regulating both the rate at which processes move through a counting network and *global* inter-operation delays. We 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 involved only such timing conditions [HSW96, LSST99, MPT97]. So, previously known results (especially the necessary conditions) apply also to sequential consistency.

- We identify timing conditions that *can* distinguish linearizability from sequential consistency; that is, these timing conditions are sufficient for sequential consistency but not for linearizability (Theorem 4.1). These conditions involve a bound on *local* inter-operation delay; thus, they are straightforward to implement. By way of example, we present, for any given *uniform* counting network,[§] timing conditions under which the network is sequentially consistent but not linearizable.

- The *fraction* of *non-sequentially-consistent* (resp., *non-linearizable*) operations in a finite execution is defined to be the ratio of the number of operations whose removal yields a sequentially consistent (resp., linearizable) execution over the total number of completed operations in the execution; so, these fractions measure the amount of locally (resp., globally) observable inconsistencies. We present both upper and lower bounds on these *inconsistency fractions* (Theorems 5.4 and 5.11).

  In particular, some of these bounds imply that in the *worst* case, some weak timing conditions, previously shown to admit a large fraction of incorrect (non-linearizable) operations [LSST99], actually admit the same fraction of non-sequentially consistent operations. Some other bounds indicate the relative cost-effectiveness of provided timing conditions for sequential consistency and linearizability.

  To derive our lower bounds on inconsistency fractions, we investigate the topological structure of (uniform) counting networks. More specifically, we identify structure with a direct quantitative impact on inconsistency fractions.

1.3 Related Work

Counting networks made up of balancers with fan-in and fan-out two were first introduced and studied in [AHS94]; those were the *bitonic* and *periodic* counting networks. A generalization

[§]Roughly speaking, a counting network is *uniform* if all paths traversing it have the same length.

was introduced in [AA95], where topological constraints on designs using larger balancers were investigated. Similar design issues were considered in [BHM94, BH02, BM98, FLL93, HKM93].

The notions of *linearizability* and *sequential consistency* were originally defined in [HW90] and [L79], respectively. Linearizable counting networks were defined and studied in [HSW96]. We know of no previous work on sequentially consistent counting networks.

To circumvent the impossibilty of a completely asynchronous linearizable counting network (with finite depth) already established in [HSW96], the first work to investigate the effect of timing conditions on the behavior of counting networks, and to identify timing conditions guaranteeing linearizability, is by Lynch *et al.* [LSST99]. Moreover, this work shows that this sufficient condition is also a necessary condition for the special cases of the bitonic counting network [AHS94] and the *counting tree* [SZ96]. Additional results in this direction are presented in [MPT97].

Generalizations to counting networks were recently considered in [BMS05, FH04]. Counting networks with additional properties (such as self-stabilization and adaptivity) have been investigated in [HT06a, T05]. Results on concurrent counting (without using counting networks) are reported in [BMT02, MT97, MTY96]. The impact of timing conditions on the relative costs of implementing linearizability and sequential consistency in message-passing has been studied in [AW94, EM99, MR99].

1.4 Road Map

Section 2 introduces our theoretical framework. Section 3 identifies timing conditions that do not distinguish linearizability from sequential consistency. Section 4 introduces timing conditions that do distinguish linearizability from sequential consistency. Our bounds on inconsistency fractions are presented in Section 5. We conclude, in Section 6, with some open problems.

2 Framework

Balancers and balancing networks are presented in Section 2.1. Section 2.2 introduces executions and counting networks. Section 2.3 proceeds to define several timing parameters and timing conditions. Definitions for consistency conditions are provided in Section 2.4. Some structural aspects of balancing and counting networks are reviewed in Section 2.5. Section 2.6 revisits some known constructions of counting networks. Some implementation issues for counting networks are discussed in Section 2.7. Section 2.8 concludes with a review of some previous, directly related work.

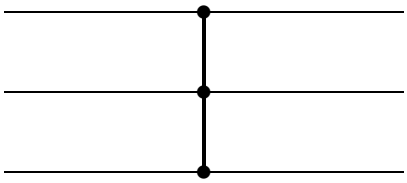

Figure 1: A symbolic representation of a $(3, 3)$-balancer.

2.1 Balancers and Balancing Networks

Many of the definitions in this section are adapted from [AA95, AHS94, LSST99].

Throughout, we consider a distributed system with an unbounded number of ***processes***. Balancing networks are constructed from elements called ***balancers***, which direct ***tokens*** from inputs to outputs, and ***wires***, which acyclically interconnect the balancers [AHS94]. The ***wires*** act as interconnection and delay elements, but provide no queueing or ordering of pending tokens.

Formally, an (f_{in}, f_{out})-***balancer***, or ***balancer*** for short, is a routing element receiving tokens on f_{in} input wires $1, 2, \ldots, f_{in}$ and sending out tokens to f_{out} output wires $1, 2, \ldots, f_{out}$; the integers f_{in} and f_{out} are called the balancer's ***fan-in*** and ***fan-out***, respectively. A balancer is ***regular*** if its fan-in and fan-out are equal. The ***state*** of an (f_{in}, f_{out})-balancer is some integer $s \in \{1, \ldots, f_{out}\}$. In the ***initial state*** of an (f_{in}, f_{out})-balancer, $s = 1$. Figure 1 depicts a $(3, 3)$-balancer; we draw wires as horizontal lines with the balancer stretched vertically. 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.

We allow processes to introduce tokens on the balancer's input wires at arbitrary times; after some delay, they shepherd them instantaneously through the balancer, arriving on an output wire. Formally, we consider an instantaneous ***balancer transition step*** of the form

$$e \quad = \quad \mathsf{BAL}_p\,(T, B, i, j)\ ,$$

corresponding to a token T of process p traversing a balancer B, entering on input wire i and exiting on output wire j.

A (w_{in}, w_{out})-***balancing network***, or ***balancing network*** for short, is a directed, acyclic graph G with three kinds of nodes:

- w_{in} ***source*** nodes $X_1, X_2, \ldots, X_{w_{in}}$;

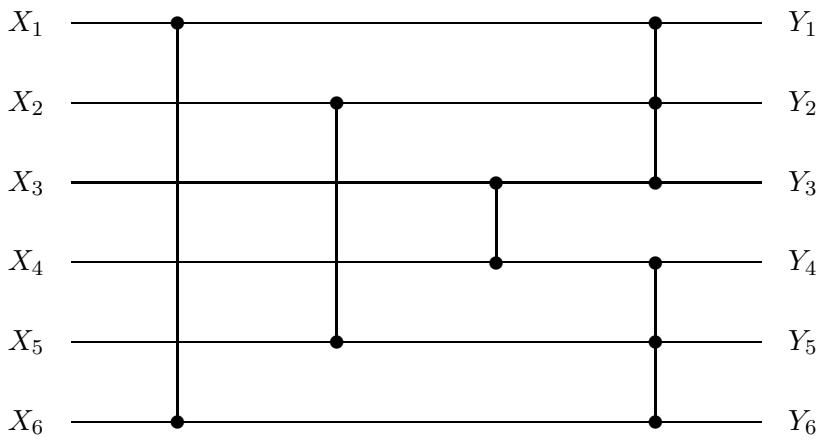

Figure 2: A $(6, 6)$-balancing network.

- w_{out} ***sink*** nodes $Y_1, Y_2, \ldots, Y_{w_{out}}$;
- a finite number of ***inner*** nodes.

The source and sink nodes represent the input and output wires of the network, respectively; the inner nodes represent the balancers of the network. The edges of G interconnect the balancers by identifying the input and output wires of them; thus, a node corresponding 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. Since edges represent wires, we shall, in the following discussion, use wires and edges interchangeably. Moreover, the outgoing and incoming degrees of all source and sink nodes, respectively, are equal to one; the incoming and outgoing degrees of all source and sink nodes, respectively, are equal to zero. The integers w_{in} and w_{out} are called the network's ***fan-in*** and ***fan-out***, respectively. When fan-in and fan-out are equal, their common value w is called the ***fan*** of the network.

We depict a balancing network as a collection of horizontal lines on which the edges are deployed; that is, each edge of the network is mapped onto some segment of a horizontal line. Balancers are stretched vertically. Figure 2 shows a $(6, 6)$-balancing network made up of $(2, 2)$-balancers and $(3, 3)$-balancers.

Each one of the w_{out} sink nodes of a balancing 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 resident counter the integers $j-1$, $(j-1)+w_{out}$, $(j-1)+2w_{out}$, and so on. The ***state*** of the counter Y_j is an integer $s \geq j-1$. In the ***initial*** state, $s = j-1$. Formally, we consider an instantaneous

counter transition step of the form

$$e \;=\; \mathsf{COUNT}_p\,(T, \mathsf{C}, v)\ ,$$

corresponding to a token T of process p traversing a counter C and obtaining the value v. A ***step*** will be used as an abbreviation for both a balancer transition step and a counter transition step.

The original definition of balancing networks [AHS94] allows each process to introduce tokens on an input wire that is either preassigned or chosen arbitrarily. In this work, we assume that each process is assigned to some specific input wire of the balancing network; all tokens generated by the process enter on that input wire. (Since the second assumption is more restrictive, it can make impossibility results only stronger.)

A ***configuration*** of a balancer is a collection of tokens on its input and output wires. A ***configuration*** of a counter is a collection of tokens on its input wire. A ***network state***, or ***state*** for short, is a collection of pairs of a state and a configuration, one pair for each of the balancers and counters in the network. In the ***initial network state***, all balancers are in their initial states and all wires are empty.

2.2 Executions and Counting Networks

An ***execution*** of a balancing network is an alternating sequence of network states and steps

$$\mathcal{E} \;=\; Q_0, e_1, Q_1, e_2, Q_2, \ldots$$

such that:

1. Q_0 is the initial network state.
2. Each step transforms the preceding network state to the following network state in the natural way. In particular:
   (a) If the step is a balancer transition step which occurs at an (f_{in}, f_{out})-balancer, then the state of the balancer increases by 1 modulo f_{out}; so, balancer states "wrap around".
   (b) If the step is a counter transition step which occurs at the counter Y_j, where $1 \leq j \leq w_{out}$, then the state of the counter increases by w_{out}; so, counter states do *not* "wrap around".

3. For each process p, for each pair of tokens T and T' issued by process p, either all steps involving T precede all steps involving T' or all steps involving T' precede all steps involving T in $\mathcal{E}$; roughly speaking, a process may not issue tokens which are overlapping in time.

4. Fix any network state Q_r, where $r \geq 1$. Use the prefix of $\mathcal{E}$ ending with Q_r to define ***history variables*** for each (f_{in}, f_{out})-balancer as follows:

   > For each index i, $1 \leq i \leq f_{in}$, the history variable x_i stands for the number of tokens that have entered on input wire i in the prefix of $\mathcal{E}$ ending with Q_r; for each index j, $1 \leq j \leq f_{out}$, the history variable y_j stands for the number of tokens that have exited on output wire j in the prefix of $\mathcal{E}$ ending with Q_r.

   Then, the following properties hold for each (f_{in}, f_{out})-balancer:

   (a) ***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.

   (b) ***Liveness property***: Assume that in the (possibly infinite) suffix of $\mathcal{E}$ starting with Q_r, there are only finitely many steps involving the balancer; then, there is a configuration $Q_{r'}$ in $\mathcal{E}$ with $r' \geq r$ where $\sum_{i=1}^{f_{in}} x_i = \sum_{j=1}^{f_{out}} x_j$; that is, a balancer never "swallows" tokens.

   Call $Q_{r'}$ a ***quiescent network state*** for the (f_{in}, f_{out})-balancer. So, in a quiescent network state $Q_{r'}$ for a balancer, the number of tokens that exited the balancer is equal to the number of tokens that entered it (in the prefix of $\mathcal{E}$ ending with $Q_{r'}$).

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

Denote by $T(\mathcal{E})$ the set of tokens appearing in execution $\mathcal{E}$. The safety and liveness properties for a balancing network follow naturally from those for its balancers and are as follows:

1. ***Safety property***: For each network state Q_r, $\sum_{i=1}^{w_{in}} x_i \geq \sum_{j=1}^{w_{out}} y_j$; that is, a balancing network never creates tokens spontaneously.

2. ***Liveness property***: Fix a network state Q_r such that in the (possibly infinite) suffix of $\mathcal{E}$ starting with Q_r, there are only finitely many steps involving each balancer; then, there is a network state $Q_{r'}$ in $\mathcal{E}$ with $r' \geq r$ which is a quiescent network state for each balancer.

   Call $Q_{r'}$ a ***quiescent network state*** for the network, or a ***quiescent network state*** for short. So, in a quiescent network state $Q_{r'}$, the number of tokens that exited the

network is equal to the number of tokens that entered it (in the prefix of $\mathcal{E}$ ending with $Q_{r'}$). Roughly speaking, a balancing network never "swallows" tokens.

We are now ready to state the most prominent correctness condition for balancing networks. A (w_{in}, w_{out})-***counting network*** [AHS94], or ***counting network*** for short, is a (w_{in}, w_{out})-balancing network which satisfies the step property:

> ***Step property***: In any execution $\mathcal{E}$, for any quiescent network state of $\mathcal{E}$, and for any pair of indices j and k such that $1 \leq j < k \leq w_{out}$, $0 \leq y_j - y_k \leq 1$.

2.3 Timing Parameters and Timing Conditions

A ***timed execution*** $R_{\mathcal{E}}$ for an execution $\mathcal{E}$ of a balancing network G is an alternating sequence of network states and ***timed steps***

$$R_{\mathcal{E}} \quad = \quad Q_0, \langle e_1, t_1 \rangle, Q_1, \langle e_2, t_2 \rangle, Q_2, \ldots$$

that associates a time t_r with each step e_r in the execution $\mathcal{E}$ in a non-decreasing order. Moreover, if the execution $\mathcal{E}$ is infinite, then the sequence $t_1, t_2, \ldots$ is unbounded.

A timed execution $R_{\mathcal{E}}$ determines a ***schedule*** $S_{\mathcal{E}} : T(\mathcal{E}) \times [\mathsf{d}(G) + 1] \rightarrow \mathbb{R}$ that specifies for any pair of a token $T \in T(\mathcal{E})$ and a layer ℓ, $1 \leq \ell \leq \mathsf{d}(G) + 1$, the time $S_{\mathcal{E}}(T, \ell)$ (as a real number) at which token T passes through a node in layer ℓ. We shall sometimes abuse notation by representing a schedule as a sequence of timed steps $R_{\mathcal{E}} = \langle e_1, t_1 \rangle, \langle e_2, t_2 \rangle, \ldots$; further, we shall interchangeably use the terms of timed execution and schedule when no confusion arises.

Associated with a schedule $S_{\mathcal{E}}$ of the 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 generated 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 $\ell - 1$, where $1 < \ell \leq \mathsf{d}(G) + 1$. 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 $\ell - 1$, where $1 < \ell \leq \mathsf{d}(G) + 1$. Intuitively, c_{max} represents the maximum delay a token incurs over any individual wire.

$\mathbf{C_L^P}$ – ***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 $\mathsf{d}(G)+1$. Intuitively, C_L^P measures the local delay incurred between the time a token by P exits the network and the time for a new token by P to enter it again.

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

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

The timing parameters c_{min}, c_{max}, and C_g were introduced by Lynch *et al.* [LSST99], who studied their impact on linearizability properties of (uniform) counting networks.

- The timing parameters c_{min} and c_{max} are motivated by the fact that many common distributed and real-time computing elements are neither completely synchronous nor completely asynchronous (cf. [AM94]). Using the parameters c_{min} and c_{max} is sufficiently general to capture both shared memory and message passing implementations of balancers [AHS94, SZ96].
- The timing parameter C_g is motivated by real-time scheduling of tasks on multiprocessors, where service requests must respect a separation by some least amount of time. If a counter is used for issuing values that break ties for the task conflicts, it is reasonable to use the parameter C_g for modeling the pattern of counter accesses.

The timing parameters c_{min}^P, C_L^P, and C_L were previously considered by Shavit *et al.* [SUZ98], who studied the impact of local delay on global performance, but with no regard to consistency conditions.

- The parameter C_L^P models the extent to which one process P issues operations at a faster rate than others; so, using these parameters allows modeling a heterogeneous distributed

system where individual processes undergo non-uniform and unpredictable delays. Finally, C_L represents the slowest possible rate of counter accesses by the processes. So, it models the process of slowest pace in a system.

A ***timing condition*** for the balancing network G is a restriction on (some combination of) the timing parameters for its schedules. A timing condition is often identified with the set of restricted timing parameters. Induced by a timing condition is a set of timed executions of the network that satisfy the condition.

2.4 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. For any pair of tokens T and T' in $T(\mathcal{E})$, say that T ***completely precedes*** T' in the execution $\mathcal{E}$ if the latest transition step involving T in $\mathcal{E}$ precedes the earliest transition step involving T' (in $\mathcal{E}$). An execution $\mathcal{E}$ specifies a partial order $\xrightarrow{\mathcal{E}}$ on tokens in $T(\mathcal{E})$ as follows:

> For any pair of tokens T and T' in $T(\mathcal{E})$, $T \xrightarrow{\mathcal{E}} T'$ if and only if T completely precedes T' in the execution $\mathcal{E}$.

A ***linearization*** of the execution $\mathcal{E}$ is a serialization of $\mathcal{E}$ that extends $\xrightarrow{\mathcal{E}}$. So, for any pair of tokens T and T' in $T(\mathcal{E})$, if $T \xrightarrow{\mathcal{E}} T'$, then T precedes T' in the linearization. An execution $\mathcal{E}$ 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*** if every execution of it is linearizable. (This is an adaptation by Herlihy *et al.* [HSW96] of the general definition of linearizability from [HW90] to balancing networks.)

Since we shall consider balancing networks in association to timing conditions, which may restrict the set of possible executions, we need to incorporate timing conditions into the standard definition of linearizability, which involves executions. Towards this end, say that a timed execution $R_{\mathcal{E}}$ is ***linearizable*** if the underlying execution $\mathcal{E}$ is linearizable. A balancing network is ***linearizable under a timing condition*** $\mathcal{C}$ if every timed execution satisfying $\mathcal{C}$ is linearizable.

We now adapt the weaker consistency condition of sequential consistency from [L79] to balancing networks. Say that an execution $\mathcal{E}$ 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*** if every execution of it is sequentially consistent. Say that a timed

execution $R_{\mathcal{E}}$ is sequentially consistent if the underlying execution $\mathcal{E}$ is sequentially consistent. A balancing network is ***sequentially consistent under a timing condition*** $\mathcal{C}$ if every timed execution satisfying $\mathcal{C}$ is sequentially consistent.

For any execution $\mathcal{E}$, consider the ***restriction*** of $\mathcal{E}$ to steps involving process P, denoted as $\mathcal{E} \mid P$. Clearly, this restriction inherits the order of tokens of 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. Our definitions immediately imply:

Observation 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.*

Fix now a balancing network G and consider a timing condition $\mathcal{C}$. Say that $\mathcal{C}$ ***distinguishes sequential consistency from linearizability for the network*** G when G is sequentially consistent under $\mathcal{C}$, but G is not linearizable under $\mathcal{C}$; call such a condition $\mathcal{C}$ a ***distinguishing timing condition*** (for the network G). Say that $\mathcal{C}$ ***does not distinguish sequential consistency from linearizability for the network*** G when G is sequentially consistent under $\mathcal{C}$ if and only if G is linearizable under $\mathcal{C}$; call such a condition $\mathcal{C}$ an ***indistinguishing timing condition*** (for the network G).

2.5 Structural Parameters and Properties of Balancing Networks

A ***path*** in a balancing network is an alternating sequence of nodes and edges, starting and finishing with a node, such that for any pair of consecutive nodes, the balancer that corresponds to the preceding node has an output wire identified with an input wire of the balancer that corresponds to the following node; these common output and input wires are represented by the edge between them in the sequence. We observe that in a counting network, there is a path from every input wire to every output wire. (To see this, note that the step property must hold even if all tokens enter the network through the same input wire.) This property will be useful in some later proofs.

A balancing network is ***uniform*** [LSST99, Definition 2.1] if each node of the network lies on some path from a source node to a sink node, and all paths from source nodes to sink nodes have the same length. We remark that all known constructions of counting networks [AA95, AVY94, AHS94, BHM94, BM98, FLL93, HKM93, KP92, SZ96] are uniform, except for the construction in [BH02].

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 as $\mathsf{d}(z)$, is defined to be zero if z is an input wire connected to a source node, and the length of the longest path from a source node to the edge otherwise. For any balancer B in a balancing network, the ***depth*** of B, denoted $\mathsf{d}(B)$, is the maximum wire depth, over all of its output wires. A ***layer*** in a balancing network is a maximal set of nodes (balancers or sinks) that have the same depth. For any integer ℓ, $1 \leq \ell \leq \mathsf{d}(G)+1$, the layer ℓ of G is the collection of nodes whose depth is ℓ. The ***depth*** of a balancing network G, denoted as $\mathsf{d}(G)$ or d for short, is the maximum balancer depth, over all of its balancers.

Fix now some output wire Y of the balancing network G. By definition of depth, there is a path π with $\mathsf{d}(G)$ wires from a source node to Y. Since there are $\mathsf{d}(G)+1$ layers, it follows that for each layer ℓ, π includes a node from layer ℓ. This implies that for each layer, for each output wire, there is a path from some node in the layer to the output wire.

2.6 Constructions of Counting Networks

We now describe some prominent constructions of counting networks, which we shall refer to later. In the descriptions, we adopt the convention that subscripts denote identical copies of the same network; we also assume that w is a power of 2. We shall start with the two original constructions of counting networks due to Aspnes *et al.* [AHS94]; the third one is one due to Shavit and Zemach [SZ96].

2.6.1 Bitonic Counting Network

The ***bitonic counting network*** $B(w)$ [AHS94] with fan w is constructed inductively. In the basis case where $w = 2$, $B(2)$ is a single $(2,2)$-balancer. In the induction step, $B(w)$ consists of two stages:

- The first stage consists of two bitonic counting networks $B_1\left(\frac{w}{2}\right)$ and $B_2\left(\frac{w}{2}\right)$ connected in parallel to the second stage.
- The second stage consists of the ***merging network*** $M(w)$ with fan w, which is constructed inductively. In the basis case, $M(2)$ is a single $(2,2)$-balancer. In the induction step, $M(w)$ consists of a column of $(2,2)$-balancers connected to two merging networks $M_1\left(\frac{w}{2}\right)$ and $M_2\left(\frac{w}{2}\right)$. The two output wires of each balancer in the row are connected to the two merging networks $M_1\left(\frac{w}{2}\right)$ and $M_2\left(\frac{w}{2}\right)$, respectively; the two input wires of each balancer in the column are connected from the two bitonic networks $B_1\left(\frac{w}{2}\right)$ and $B_2\left(\frac{w}{2}\right)$,

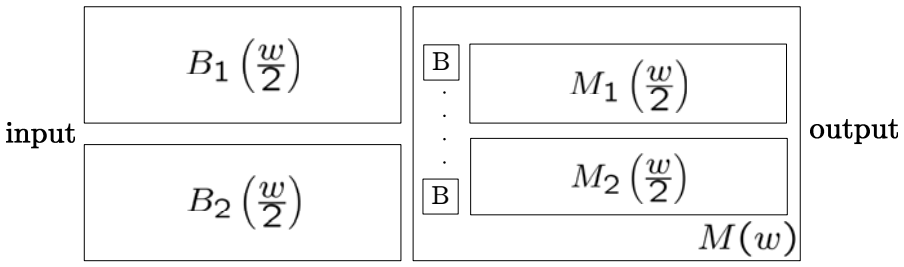

Figure 3: The bitonic counting network $B(w)$ in diagrammatic form. We have denoted balancers using the symbol B. (Balancers and wires have been omitted.)

respectively; Note that the network $M(w)$ has depth $\mathsf{d}(M(w)) = \lg w$. We remark that there is a path from each input wire to each output wire of the network $M(w)$.

The depth of the bitonic counting network is $\mathsf{d}(B(w)) = \frac{\lg w(\lg w + 1)}{2}$. The inductive construction of the bitonic counting network is depicted (in diagrammatic form) in Figure 3. As a particular example, the bitonic counting networks $B(4)$ and $B(8)$ are shown in Figure 4.

2.6.2 Periodic Counting Network

The ***periodic counting network*** [AHS94] with fan w is the cascade of $\lg w$ components. Each component is a ***block network*** $L(w)$ of fan w, which consists of two stages:

- The first stage consists of two block networks $L_1\left(\frac{w}{2}\right)$ and $L_2\left(\frac{w}{2}\right)$ connected in parallel to the second stage.
- The second stage consists of the ***odd-even network*** $OE(w)$, which consists of a single column of $(2,2)$-balancers. Each of the two input wires of each balancer in the column is identified with an output wire of the networks $L_1\left(\frac{w}{2}\right)$ and $L_2\left(\frac{w}{2}\right)$, respectively.

Alternatively, the block network can be constructed to consist of two stages, as follows:

- The first stage consists of the ***top-bottom*** network $TB(w)$ with fan w, which consists of a single column of $(2,2)$-balancers. The two input (resp., output) wires of each balancer in the column are located symmetrically (with respect to the middle) in the sequences $1,\ldots,\frac{w}{2}$ and $\frac{w}{2}+1,\ldots,w$.

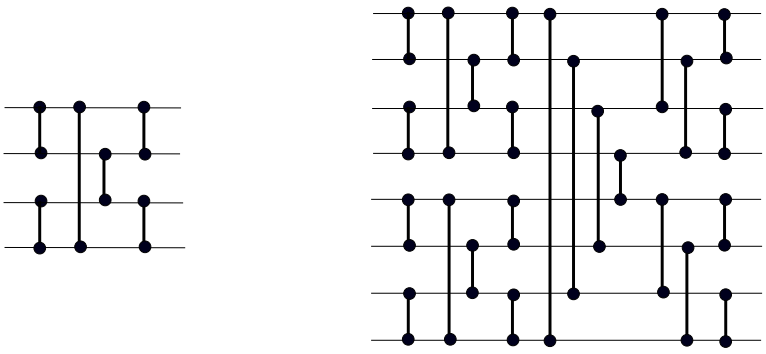

Figure 4: The bitonic counting networks $B(4)$ and $B(8)$ on left and right, respectively.

- The second stage consists of a block network $L_1\left(\frac{w}{2}\right)$ and an extension $\widehat{L_2}\left(\frac{w}{2}\right)$ to a block network $L_2\left(\frac{w}{2}\right)$. The extension $\widehat{L_2}\left(\frac{w}{2}\right)$ is obtained by renaming each input (and output) wire i of the network $L_2\left(\frac{w}{2}\right)$ to $\left(i + \frac{w}{2}\right) \mod w$.

As an example, Figure 5 demonstrates the two described constructions for the network $L(8)$. Note that the network $L(w)$ has depth $\mathsf{d}(L(w)) = \lg w$. It is established by Herlihy and Tirthapura [HT06] that the block network $L(w)$ and the merging network $M(w)$ are *isomorphic* (as graphs). This verifies that there is a path from each input wire to each output wire of $L(w)$.

The depth of the periodic counting network $P(w)$ is $\mathsf{d}(P(w)) = \lg w \cdot \mathsf{d}(L(w)) = \lg^2 w$. Figure 6 depicts the periodic counting network $P(w)$ (in diagrammatic form), where the construction of each block network follows the second alternative.

2.6.3 Counting Tree

The $(w, 1)$-***counting tree***, or ***counting tree*** for short (also known as ***diffracting tree*** [SZ96]) is a balanced binary tree of depth $\lg w$ made up of $(2, 1)$-balancers.

2.7 Implementing Counting Networks

On a shared memory multiprocessor machine, a counting network is implemented as a data structure in shared memory; 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

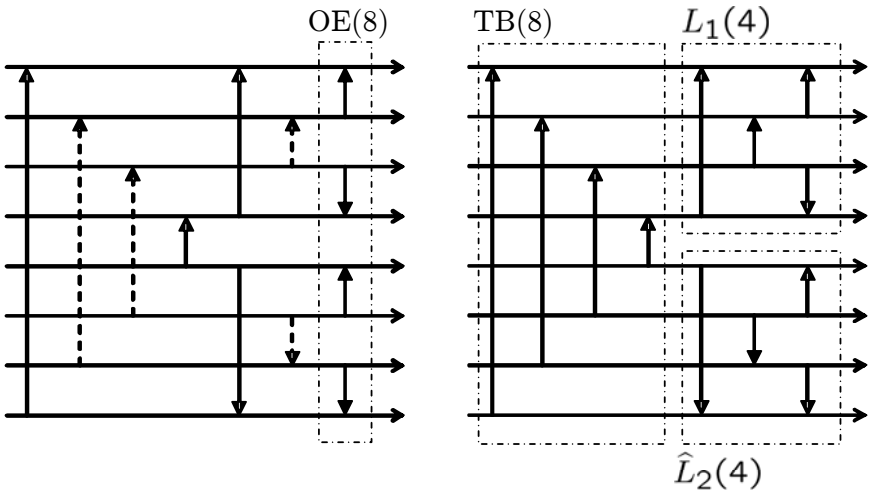

Figure 5: The block network $L(8)$ in both left and right. The left network features the first construction. Here, the block network $L(8)$ consists of two interleaved block networks $L(4)$, whose balancers are denoted by solid and dotted lines, respectively. The outputs of the two smaller block networks are fed into the boxed odd-even network $OE(8)$. The right network features the second construction. Here, the boxed top-bottom network $TB(8)$ feeds into the two boxed block networks $L_1(4)$ and $\widehat{L}_2(4)$ connected in parallel.

time it shepherds a new token through the network. In doing so, the process atomically updates each balancer and uses the returned value to choose which pointer to follow. Upon exiting the data structure, the token is pointed to a local *counter* which assigns (unique) values that are congruent modulo the fan-out of the network. (Since the balancers are connected correctly, all consecutive values $1, 2, \ldots$ will be assigned (with no gaps).)

2.8 Previous Related Results

It is pointed out in [LSST99] that the timing condition $\mathsf{d}(G) \cdot (c_{max} - 2c_{min}) < C_g$ [LSST99, Corollary 3.7] relating depth, wire delays and global delay, and which is sufficient for linearizability, is *not* a local condition – it would require coordination among individual processes in order to ensure that the lower bound on C_g is preserved. Hence, the stronger sufficient condition $\frac{c_{max}}{c_{min}} \leq 2$ [LSST99, Corollary 3.10] is stressed as a local linearizability criterion. (As we shall see in Section 4, some weaker, local timing conditions suffice to guarantee the weaker correctness condition of sequential consistency.) Table 1 summarizes all known previous results that provide necessary and sufficient timing conditions for linearizability in counting networks.

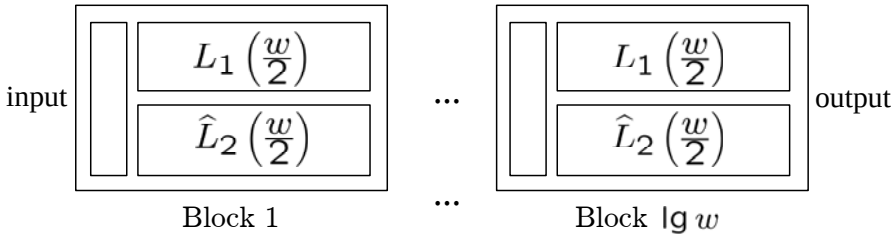

Figure 6: The periodic counting network $P(w)$ in diagrammatic form. Each of the numbered $\lg w$ boxes represents a block network $L(w)$. Inside each box, the left-most rectangle represents a top-bottom network $TB(w)$. (Balancers and wires have been omitted.)

3 Non-Distinguishing Timing Conditions

In this section, we establish that constraints that only consider the bounds c_{min}, c_{max} and C_g *cannot* distinguish sequential consistency from linearizability.

The proof of this indistinguishability result exploits the modular counting carried out by an individual balancer with fan-out f; that is, f tokens can be simultaneously carried through a balancer without affecting the values returned to later tokens. We formalize this property as follows:

Lemma 3.1 (Modular Counting) *Fix a timed execution $R_{\mathcal{E}} = \langle e_1, t_1\rangle, \ldots, \langle e_r, t_r\rangle, \widehat{R}_{\mathcal{E}}$ of a balancer B with fan-out f, where τ tokens have traversed B in the prefix of $R_{\mathcal{E}}$ ending with $\langle e_l, t_l\rangle$. Consider processes $p_1, \ldots, p_f$ assigned to input wires $i_1, \ldots, i_f$ of B, respectively, and corresponding tokens $T_1, \ldots, T_f$. Then, for any real number $t \geq t_r$ such that either $\widehat{R}_{\mathcal{E}}$ is empty or $t \leq t_{r+1}$, the sequence*

$$\begin{aligned} R_{\mathcal{E}'} \;=\; & \langle e_1, t_1\rangle, \ldots, \langle e_r, t_r\rangle, \\ & \langle \mathsf{BAL}_{p_1}(T_1, B, i_1, \tau \bmod f+1), t\rangle, \ldots, \\ & \langle \mathsf{BAL}_{p_f}(T_f, B, i_f, \tau \bmod f+f), t\rangle, \\ & \widehat{R}_{\mathcal{E}} \end{aligned}$$

is also a timed execution of B.

Roughly speaking, Lemma 3.1 deals with a timed execution of a balancer B; it considers inserting f steps by processes $p_1, \ldots, p_f$ assigned to input wires $x_1, \ldots, x_f$, with corresponding tokens $T_1, \ldots, T_f$; the insertion takes place immediately following the step $\langle e_r, t_r\rangle$ in $R_{\mathcal{E}}$. These additional steps occur simultaneously at time t; since times are non-decreasing in a timed

Counting networks	Sufficient condition	Necessary condition
Arbitrary	$\frac{c_{max}}{c_{min}} \leq \frac{2\mathsf{s}(G)}{\mathsf{d}(G)}$ [MPT97, Thm. 4.1]	—
Uniform	$\mathsf{d}(G)(c_{max} - 2c_{min}) < C_g$ [LSST99, Cor. 3.7]	$\frac{c_{max}}{c_{min}} \leq \frac{\mathsf{d}(G)}{\mathsf{irad}(G)} + 1$ [MPT97, Thm. 3.1]
	$\frac{c_{max}}{c_{min}} \leq 2$ [LSST99, Cor. 3.10]	
Bitonic	$\downarrow$	$\frac{c_{max}}{c_{min}} \leq 2$ [LSST99, Thm. 4.3]
Counting Tree	$\downarrow$	$\frac{c_{max}}{c_{min}} \leq 2$ [LSST99, Thm. 4.1]

Table 1: Known necessary and sufficient timing conditions for linearizability in counting networks. A downward arrow $\downarrow$ indicates that the sufficient condition is identical to the sufficient condition(s) for the immediately wider class of counting networks in the table. So, the sufficient conditions for uniform counting networks are inherited down to both the bitonic counting network and the tree. A bar indicates lack of known results. Here, for a counting network G, $\mathsf{irad}(G)$, called the *influence radius* of G, denotes the maximum, over all pairs of output wires j and k of G, of the distance from j to the least common ancestor of j and k in G; $\mathsf{s}(G)$, called the *shallowness* of G, denotes the length of the shortest path from an input wire to an output wire of the network. (Clearly, $\mathsf{s}(G) \leq \mathsf{d}(G)$, while $\mathsf{s}(G) = \mathsf{d}(G)$ if and only if G is uniform.)

execution, it must be that $t \geq t_r$; moreover, if the suffix $R_{\mathcal{E}}$ is not empty, t may not exceed the time t_{r+1} at which the step following $\langle e_l, t_l \rangle$ occurs in $R_{\mathcal{E}}$. Since τ tokens have traversed B in the prefix of $R_{\mathcal{E}}$ ending with the step $\langle e_l, t_l \rangle$, the new tokens must exit through the output wires $\tau \bmod f + 1, \ldots, \tau \bmod f + f$, where additions are modulo f; this follows immediately from the observation that a balancer with fan-out f acts as a counter modulo f. Since exactly f new tokens pass through B simultaneously, the state of the balancer returns to its original value; so, the outgoing wire to be used next is not affected. We are now ready to prove:

Theorem 3.2 (Non-Distinguishing Timing Condition) *For a uniform counting network, there is no timing condition c_{min}, c_{max} and C_g that distinguishes sequential consistency from linearizability.*

Proof: Fix a uniform counting network G with fan-out W and depth d. Assume, by way of contradiction, that there is a timing condition c_{min}, c_{max} and C_g that distinguishes sequential consistency from linearizability. So, fix a timed execution $R_{\mathcal{E}}$ of G that satisfies the timing condition but is not linearizable. We will use $R_{\mathcal{E}}$ to construct another timed execution $R'_{\mathcal{E}}$ of G that is not sequentially consistent and yet satisfies the same timing condition.

For simplicity, assume for now that each balancer in the network G is regular. We will later extend our construction to remove this assumption.

Since $R_{\mathcal{E}}$ is not linearizable, there are two tokens T and T', such that T completely precedes T' in $\mathcal{E}$, while T and T' return values v and v' such that $v > v'$.

- If these two tokens are introduced by the same process, then $R_{\mathcal{E}}$ is already not sequentially consistent, and we are done. So, assume that T and T' are introduced by different processes p and p', respectively.
- If each process may introduce tokens on arbitrarily chosen input wires, the claim follows trivially by relabeling tokens T and T' to be introduced by a new process that otherwise takes no steps in $R_{\mathcal{E}}$. So, assume that this is not the case; denote as i and i' the (possibly different) input wires on which tokens T and T' are introduced by the two processes p and p'.

We start with an informal outline of our proof. We will carefully introduce and schedule additional tokens by using the modular counting property of balancers from Lemma 3.1. In this way, we will obtain two tokens associated with the same process (and introduced on the same input wire) that mimic the behavior of tokens T and T', emerging with the values v and v', respectively. We now continue with the details of the formal proof.

Since the counting network G with fan-out W is uniform and all balancers are regular, it follows that the fan-in of the network is equal to W as well. So, consider a new set of processes $p_1, \ldots, p_W$ that take no steps in the execution $R_{\mathcal{E}}$; assume that each such process p_l is assigned to the input wire l, $1 \leq l \leq W$. Our construction can be divided into two main parts; we start with the first.

- Relabel the process indices of all steps of token T from p to p_i.

Denote by $\widehat{R}_{\mathcal{E}}$ the resulting timed execution. Clearly, $\widehat{R}_{\mathcal{E}}$ satisfies the timing condition c_{min}, c_{max} and C_g. In addition, each token receives the same value in $\widehat{R}_{\mathcal{E}}$ as in $R_{\mathcal{E}}$. In particular, token T receives the value v in the timed execution $\widehat{R}_{\mathcal{E}}$. Write

$$\begin{aligned}\widehat{R}_{\mathcal{E}} \;=\; & \langle e_1, t_1\rangle, \ldots, \\ & \langle \mathsf{BAL}_q(T', B_1, i_1, j_1), t_{q_1}\rangle, \ldots, \\ & \langle \mathsf{BAL}_q(T', B_{\mathsf{d}-1}, i_{\mathsf{d}-1}, j_{\mathsf{d}-1}), t_{q_{\mathsf{d}-1}}\rangle, \ldots, \\ & \langle \mathsf{COUNT}_q(T', \mathsf{C}_{\mathsf{d}}, v_2), t_{q_{\mathsf{d}}}\rangle, \ldots\ ,\end{aligned}$$

where the identified steps are the d steps taken by process p' to move the token T' through the network and access the counter $\mathsf{C_d}$.

We now continue with the second part of our construction. In this part, we will use a path π from input wire i to the counter $\mathsf{C_d}$. (Recall that such a path must exist.) Using Lemma 3.1, we will carefully route a token by process p_i along π, emerging just before T' and returning the value v'. In doing so, we must prevent this token from affecting other tokens. To achieve this, we will use W additional tokens, which will be routed synchronously through the uniform network, moving through each layer at the same speed as T'. These W additional tokens will suffice to guarantee that exactly one additional token will arrive on each input wire of a balancer in the network. Hence, exactly one of the additional tokens will exit on the output wire of each balancer, without affecting the state of the balancer. Thus, subsequent tokens will not be affected. We continue with the formal details of the second part of the construction.

- Insert W steps immediately preceding the step $\langle \mathsf{BAL}_q(T', B_1, i_1, j_1), t_{q_1} \rangle$ in the timed execution $\widehat{R}_{\mathcal{E}}$. Each such step corresponds to a token T_{p_l} by process p_l, $1 \leq l \leq W$, entering the network through its assigned input wire l. All such steps occur at time t_{q_1}. The ordering of the inserted steps is such that the token T_{p_i} introduced by process p_i is routed through the first balancer B_1 on the path π (to the counter $\mathsf{C_d}$). Denote as $R_{\mathcal{E}_1}$ the resulting timed sequence.

So, one token has moved from every input wire through the first balancer B_1 on the path π. Hence, there is now one token on each input wire of the second layer. We now repeat:

- Insert W steps immediately preceding the step $\langle \mathsf{BAL}_q(T', B_2, i_2, j_2), t_{q_2} \rangle$ in the timed execution $\widehat{R}_{\mathcal{E}_1}$. Each such step corresponds to a token T_{p_l} by process p_l, $1 \leq l \leq W$, entering the second layer of the network. All such steps occur at time t_{q_2}. The ordering of the inserted steps is such that the token introduced by process p_i is routed through the second balancer on the path π (to the counter $\mathsf{C_d}$). Denote as $R_{\mathcal{E}_2}$ the resulting timed sequence.
- We repeat this insertion procedure for a total of $\mathsf{d} - 1$ times. Denote as $R_{\mathcal{E}_{\mathsf{d}-1}}$ the resulting timed sequence.

Clearly, in the timed sequence $R_{\mathcal{E}_{\mathsf{d}-1}}$, the token T_{p_i} is on the input wire to the counter $\mathsf{C_d}$. We are now ready for the last step of our construction:

- $R_{\mathcal{E}_{\mathsf{d}}} := R_{\mathcal{E}_{\mathsf{d}-1}}, \langle \mathsf{COUNT}_{p_i}(T_{p_i}, C_{\mathsf{d}}, v), t_{q_{\mathsf{d}}} \rangle$. (The value v will be determined later.)

Note that for each index ℓ, $1 \leq \ell \leq \mathsf{d}-1$, the restriction of the timed sequence $R_{\mathcal{E}_\ell}$ to each individual balancer and counter of G is a timed execution; hence, $R_{\mathcal{E}_\ell}$ is a timed execution of the network G. This implies that $R_{\mathcal{E}_\mathsf{d}}$ is a timed execution of the network G.

Note that in the timed execution $R_{\mathcal{E}_\mathsf{d}}$, each of the W additional tokens traverses each layer of the network G at exactly the same rate as the token T' in the original timed execution $R_\mathcal{E}$; each of the older tokens keeps its own rate. Moreover, the time between any two non-overlapping tokens in $R_{\mathcal{E}_\mathsf{d}}$ is the same as the time between two non-overlapping tokens in $R_\mathcal{E}$. It follows that the timed execution $R_{\mathcal{E}_\mathsf{d}}$ satisfies the timing condition c_{min}, c_{max} and C_g.

Consider any index ℓ, $1 \leq \ell \leq \mathsf{d}-1$. Since there are W additional tokens all balancers are regular and the network is uniform, it follows that the number of additional tokens taking a step through each balancer is equal to the fan of the balancer. Hence, Lemma 3.1 implies that all balancers are left in the same state in the timed execution $R_{\mathcal{E}_\ell}$ as they were in the corresponding prefix of the timed execution $R_\mathcal{E}$ ending with the step $\langle \mathsf{BAL}_q(T_2, B_\ell, i_\ell, j_\ell), t_{q_\ell}\rangle$. Since none of the additional tokens traversed a counter in the timed execution $R_{\mathcal{E}_\ell}$, this implies that all counters are also left in the same state in the timed execution $R_{\mathcal{E}_\ell}$ as they were in the corresponding prefix of the timed execution $R_\mathcal{E}$ ending with the step $\langle \mathsf{BAL}_q(T_2, B_\ell, i_\ell, j_\ell), t_{q_\ell}\rangle$. In particular, all balancers and counters are left in the same state in the timed execution $R_{\mathcal{E}_{\mathsf{d}-1}}$ as they were in the corresponding prefix of the timed execution $R_\mathcal{E}$ ending with the step $\langle \mathsf{BAL}_q(T_2, B_{\mathsf{d}-1}, i_{\mathsf{d}-1}, j_{\mathsf{d}-1}), t_{q_{\mathsf{d}-1}}\rangle$. It follows that the token T_{p_i} receives the same value in the timed execution $R_{\mathcal{E}_\mathsf{d}}$ as the token T' receives in the timed execution $R_\mathcal{E}$; so, $v = v'$. Since T completely precedes T_{p_i}, this implies that $R_{\mathcal{E}_\mathsf{d}}$ is not sequentially consistent.

So, in conclusion, there is a timed execution $R_{\mathcal{E}_\mathsf{d}}$ of the network G, satisfying the timing condition c_{min}, c_{max} and C_g, which is not sequentially consistent. A contradiction.

So far we have assumed that all balancers are regular. We now describe an extension to our construction that removes this assumption; the resulting construction may need far more than W tokens. Specifically, for each layer ℓ of the network G, where $1 \leq \ell \leq \mathsf{d}-1$, denote as LCM_ℓ the *least common multiple* of the fan-outs of balancers in layer ℓ of G. Set $\mathsf{LCM} = \prod_{\ell=1}^{\mathsf{d}-1} \mathsf{LCM}_\ell$. So, LCM is a multiple of each balancer's fan-out.

- Construct the timed execution $R_{\mathcal{E}_\mathsf{d}}$ by initially entering LCM tokens on each input wire of the network G. These tokens will be simultaneously traversing each layer of the network, as in the original construction.

Clearly, at least one token will emerge on each wire of the network; moreover, the number of tokens traversing each balancer is a multiple of its fan-out, as required by Lemma 3.1. So, this extension suffices to route the specific token T_{p_i} to counter C_d, and all arguments still apply in

an identical way to yield that there is a timed execution $R_{\mathcal{E}_\mathsf{d}}$ of the network G, satisfying the timing condition c_{min}, c_{max} and C_g, which is not sequentially consistent. A contradiction. ■

An inspection to the proof of Theorem 3.2 reveals that the proof would become trivial under the more general assumption that each process can introduce tokens on arbitrarily chosen input wires.

The results reported in [HSW96, LSST99, MPT97] 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. Thus, together Theorem 3.2 and the results proved for linearizability in [LSST99, MPT97] (see Table 1) immediately imply:

Corollary 3.3 *A uniform counting network G is sequentially consistent under timing condition c_{min} and c_{max} only if*

$$\frac{c_{max}}{c_{min}} \leq \frac{d(G)}{\mathsf{irad}(G)} + 1\,.$$

Corollary 3.4 *A bitonic counting network (resp., counting tree) is sequentially consistent under timing condition c_{min} and c_{max} if and only if*

$$\frac{c_{max}}{c_{min}} \leq 2\,.$$

Notice that the local delay C_L is not constrained by any of the necessary and sufficient conditions for linearizability and sequential consistency shown in this section. Note, however, that for an arbitrary uniform counting network G, Corollary 3.3 implies that even for some small enough local delay (say 0), G is not sequentially consistent; in the next section, we will formally prove (Theorem 4.1) that, in contrast, if the local delay is large enough, then G *is* sequentially consistent.

4 Distinguishing Timing Conditions

In this section, we establish that any uniform counting network G is sequentially consistent under a timing condition c_{min}, c_{max} and C_L such that $\mathsf{d}(G) \cdot (c_{max} - 2c_{min}) < C_L$, but that this condition is insufficient for linearizability. However, unlike the global delay bound $\mathsf{d}(G) \cdot (c_{max} - 2c_{min}) < C_g$, (which implies linearizability [LSST99, Corollary 3.7]) this condition can

be implemented easily using local clocks: upon completion of an operation by a process, the process sets a timer to expire after time $\mathsf{d}(G) \cdot (c_{max} - 2c_{min})$ elapses; it may then issue another operation. We first prove:

Theorem 4.1 (Sufficient Condition for Sequential Consistency) *Fix a uniform counting network G. Consider a timing condition c_{min}, c_{max} and C_L such that $\mathsf{d}(G) \cdot (c_{max} - 2c_{min}) < C_L$. Then, G is sequentially consistent under this condition.*

To prove Theorem 4.1, we use the following technical claim due to Lynch *et al.* [LSST99].

Proposition 4.2 ([LSST99]) *Consider arbitrary tokens T and T' traversing a uniform counting network G during the time intervals $[t_{in}, t_{out}]$ and $[t'_{in}, t'_{out}]$, respectively. Assume that $\mathsf{d}(G) \cdot (c_{max} - 2c_{min}) < t'_{in} - t_{out}$. Then, T' returns a higher value than T.*

Proposition 4.2 can be restricted to the case where tokens T and T' are introduced by the same process P to yield:

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

We now use Corollary 4.3 to prove:

Lemma 4.4 *Fix a uniform counting network G and a process P. Consider a timing condition c^P_{min}, c_{max} and C^P_L such that $\mathsf{d}(G) \cdot (c_{max} - 2c^P_{min}) < C^P_L$. Then, G is sequentially consistent for process P under this condition.*

Proof: Consider any pair of tokens T and T', both of process P, traversing G during the time intervals $[t_{in}, t_{out}]$ and $[t'_{in}, t'_{out}]$, respectively, with T preceding T'. By definition of C^P_L, it follows that $C^P_L \leq t'_{in} - t_{out}$. Hence, the assumption implies that $\mathsf{d}(G) \cdot (c_{max} - 2c^P_{min}) < t'_{in} - t_{out}$. Thus, Corollary 4.3 implies that T' returns a higher value than T. Since T and T' were chosen arbitrarily, it follows that G is sequentially consistent for process P. ■

Theorem 4.1 follows now from the definition of C_L, Lemma 4.4 and Proposition 2.1.

We now use Theorem 4.1 and Corollary 3.3 to prove that for any uniform counting network, there is a distinguishing timing condition:

Corollary 4.5 (Distinguishing Timing Condition) *Fix a uniform counting network G. Then, there is a distinguishing timing condition c_{min}, c_{max} and C_L for G.*

Proof: Consider a timing condition c_{min}, c_{max} and C_L such that *(i)* $\frac{c_{max}}{c_{min}} > \frac{\mathsf{d}(G)}{\mathsf{irad}(G)} + 1$ and *(ii)* $C_L > \mathsf{d}(G) \cdot (c_{max} - 2\,c_{min})$. We prove that G is sequentially consistent but not linearizable under this timing condition.

- By Theorem 4.1, *(ii)* suffices to imply that G is sequentially consistent under this timing condition.
- To prove that G is not linearizable under the timing condition, we determine a timed execution of G that satisfies the timing condition, yet it is not linearizable. Corollary 3.3 implies that there is a timed execution $R_{\mathcal{E}}$ of G that is not sequentially consistent and yet satisfies *(i)*. Rename processes that introduce more than one token in execution $\mathcal{E}$, so that each token is now introduced by a different process. Denote as $R'_{\mathcal{E}}$ the resulting timed execution.
  - The construction implies that $R'_{\mathcal{E}}$ still satisfies the first inequality, while it vacuously satisfies the second inequality. So, $R'_{\mathcal{E}}$ satisfies the timing condition.
  - Since $R_{\mathcal{E}}$ is not sequentially consistent, the construction implies that $R'_{\mathcal{E}}$ is not linearizable.

The proof is now complete. ■

5 Inconsistency Fractions

Section 5.1 contains our definitions for inconsistency fractions and some preliminary facts. Upper and lower bounds on inconsistency fractions are presented in Sections 5.2 and 5.3, respectively.

5.1 Definitions and Preliminaries

All definitions here refer to a fixed balancing network G; we will omit explicit mention of G.

A token T is ***non-linearizable*** [LSST99, Definition 2.5] in an execution $\mathcal{E}$ if there is some other token T' in $\mathcal{E}$, which completely precedes T in $\mathcal{E}$ and returns a value larger than that of T. The choice of declaring T as the non-linearizable token (and not T') is justified in [LSST99,

discussion following Definition 2.5] in two ways. First, it allows determining whether or not a token is non-linearizable as soon as it completes. Second, if T' were instead declared as the non-linearizable token, the definition would lead to non-reasonable situations where a single token can cause *all* tokens preceding it to become non-linearizable by returning a sufficiently small value. A token T is ***non-sequentially consistent*** in an execution $\mathcal{E}$ if there is some other token T' in $\mathcal{E}$, introduced by the same process, which precedes T in $\mathcal{E}$ and returns a value larger than that of T.

Fix now a finite execution $\mathcal{E}$ with a total number of tokens $|T(\mathcal{E})|$. The ***non-linearizability fraction*** of $\mathcal{E}$ [LSST99] is the number of non-linearizable tokens in $\mathcal{E}$ divided by $|T(\mathcal{E})|$. In a similar way, we define the ***non-sequential consistency fraction*** of $\mathcal{E}$ as the number of non-sequentially consistent tokens in $\mathcal{E}$ divided by $|T(\mathcal{E})|$.

Lynch *et al.* [LSST99, discussion following Definition 2.6] observe that the non-linearizable fraction is an *upper bound* on the fraction of tokens whose removal yields a linearizable sequence. So, define the ***absolute non-linearizability fraction*** of $\mathcal{E}$ as the *least* number of non-linearizable tokens in $\mathcal{E}$ whose removal yields a linearizable execution divided by $|T(\mathcal{E})|$. We prove:

Lemma 5.1 *For a finite execution $\mathcal{E}$, the non-linearizability fraction of $\mathcal{E}$ is equal to the absolute non-linearizability fraction of $\mathcal{E}$.*

Proof: The claim holds vacuously when $\mathcal{E}$ is linearizable. So, assume that $\mathcal{E}$ is not linearizable. Assume, by way of contradiction, that the absolute non-linearizability fraction of $\mathcal{E}$ is strictly less than the non-linearizability fraction of $\mathcal{E}$. So, it is possible to remove a strict subset of the non-linearizable tokens in $\mathcal{E}$ and obtain a linearizable execution $\mathcal{E}'$. Fix any non-linearizable token T that is not removed.

Since all linearizable tokens in $\mathcal{E}$ are also in $\mathcal{E}'$, it follows that there is no linearizable token in $\mathcal{E}$ that completely precedes T in $\mathcal{E}$ and returns a larger value than T.

So, consider all tokens that completely precede T in $\mathcal{E}$ and return larger values than T. It follows that all these tokens are non-linearizable (in $\mathcal{E}$). Take T' to be the token with the earliest step in $\mathcal{E}$. Since T' is non-linearizable in $\mathcal{E}$, it follows that there is a token T'' (in $\mathcal{E}$) that precedes T' (in $\mathcal{E}$) and returns a larger value than T'.

- Since T' has been taken to be the *earliest* token in $\mathcal{E}$ (among all non-linearizable tokens in $\mathcal{E}$ that completely precede T in $\mathcal{E}$ and return a larger value than T), it follows that T'' is linearizable.

- Since T'' completely precedes T' (in $\mathcal{E}$) and T' completely precedes T (in $\mathcal{E}$), it follows that T'' completely precedes T in $\mathcal{E}$.
- Since T'' returns a larger value than T' (in $\mathcal{E}$) and T' returns a larger value than T (in $\mathcal{E}$), it follows that T'' returns a larger value than T in $\mathcal{E}$.

A contradiction. ■

Fix now a timing condition $\mathcal{C}$. The ***non-linearizability fraction under*** $\mathcal{C}$ [LSST99], denoted as $\mathbf{F}_{nl}(G)$, is the maximum, over all (finite) timed executions $\mathcal{E}$ satisfying $\mathcal{C}$, of the non-linearizability fraction of $\mathcal{E}$. The ***non-sequential consistency fraction under*** $\mathcal{C}$. denoted as $\mathbf{F}_{nsc}(G)$, is the maximum, over all (finite) timed executions $\mathcal{E}$ satisfying $\mathcal{C}$, of the non-sequential consistency fraction of $\mathcal{E}$. Clearly, $\mathbf{F}_{nl}(G) \geq \mathbf{F}_{nsc}(G)$.

In the literature, there is a single known bound on inconsistency fractions; this is a lower bound on $\mathbf{F}_{nl}(G)$ for the particular case where G is the bitonic counting network [AHS94] and under a timing condition involving the fan of the network. The following result is due to Lynch *et al.* [LSST99, Theorem 4.4]:

Proposition 5.2 ([LSST99]) *Consider the bitonic counting network $B(w)$ with fan w, under a timing condition c_{min} and c_{max} such that $\frac{c_{max}}{c_{min}} > \frac{\lg w + 3}{2}$. Then,*

$$\mathbf{F}_{nl}(B(w)) \quad \geq \quad \frac{1}{3}\,.$$

We modify the construction used in the proof of Proposition 5.2 to get a slightly stronger result:

Proposition 5.3 *Consider the bitonic counting network $B(w)$ with fan w, under a timing condition c_{min} and c_{max} such that $\frac{c_{max}}{c_{min}} > \frac{\lg w + 3}{2}$. Then,*

$$\mathbf{F}_{nsc}(B(w)) \quad \geq \quad \frac{1}{3}\,.$$

The proof of Proposition 5.3 will refer to the construction of the bitonic counting network, as illustrated in Figure 3.

Proof: We construct a (finite) timed execution $R_{\mathcal{E}}$ of the network $B(w)$ with three *waves* of tokens; each wave has $\frac{w}{2}$ tokens which enter the network simultaneously and proceed in lock step.

- The first wave enters through the network $B_1(\frac{w}{2})$. Since $B(w)$ is a counting network, this wave proceeds to the network $M_1(\frac{w}{2})$. The speed of this wave in the network $M(w)$ is the slowest– one wire per time c_{max}.

- Immediately following the first wave (and with no relative dalay) is the second wave, which also enters through the network $B_1(\frac{w}{2})$. Since $B(w)$ is a counting network and the second wave follows the first wave (with no relative delay), this second wave proceeds to the network $M_2(\frac{w}{2})$. The speed of this wave in the network $M(w)$ is the fastest– one wire per time c_{min}. Assume that the $\frac{w}{2}$ tokens in this wave are introduced by processes $p_1, \ldots, p_{\frac{w}{2}}$, respectively.

- As soon as the second wave exits the network, the third wave enters through the network $B_1(\frac{w}{2})$. Since $B(w)$ is a counting network and the two preceding waves proceeded to the networks $M_1(\frac{w}{2})$ and $M_2(\frac{w}{2})$,. respectively, this third wave proceeds to the network $M_1(\frac{w}{2})$. The speed of this wave in the network $M(w)$ is the fastest– one wire time per time c_{min}. Assume that the $\frac{w}{2}$ tokens in this wave are introduced by processes $p_1, \ldots, p_{\frac{w}{2}}$, respectively.

Clearly, the values returned to the tokens in the second wave are $\frac{w}{2}, \ldots, w-1$, respectively. Denote as t_1, t_2 and t_3 the times for the first, second and third waves, respectively, to reach the counters of the network; these times are taken from the point where the first wave is entering the network $M(w)$. Clearly, $t_1 = \mathsf{d}(M(w)) \cdot c_{max} = \lg w \cdot c_{max}$, $t_2 = \mathsf{d}(M(w)) \cdot c_{min} = \lg w \cdot c_{min}$ and $t_3 = t_2 + \mathsf{d}(B(w)) \cdot c_{min} = t_2 + \frac{\lg w(\lg w + 1)}{2} \cdot c_{min}$. Hence,

$$\begin{aligned} & t_3 \\ = \;& \lg w \cdot \frac{\lg w + 3}{2} \cdot c_{min} \\ < \;& \lg w \cdot c_{max} \qquad \text{(by assumption)} \\ = \;& t_1 \,. \end{aligned}$$

This implies that the third wave bypasses the first wave on the first wire out and obtains values $0, \ldots, \frac{w}{2} - 1$; these values are all smaller than the corresponding values returned to tokens by the same processes in the first wave. So, there are $\frac{w}{2}$ non-sequentially consistent tokens in the execution $\mathcal{E}$ with $\frac{3w}{2}$ tokens, and the claim follows. ■

We remark that the only difference between the proof of Proposition 5.2 from [LSST99] and the proof of Proposition 5.3 is that in the latter, the set of processes shepherding tokens in the third wave is taken to be the same as the set of processes shepherding tokens in the second wave.

Propositions 5.2 and 5.3 establish a lower bound of $\frac{1}{3}$ on the non-linearizability and non-sequential consistency fractions, respectively, for the bitonic counting network $B(w)$ under a certain timing condition. We feel that this lower bound is rather poor for practical applications. On the other hand, the required asynchrony to guarantee this lower bound on inconsistency fractions amounts to a lower bound of $\frac{\lg w + 3}{2}$ on the ratio $\frac{c_{max}}{c_{min}}$; this lower bound grows unbounded as the fan w of the network increases. This fact confirms the intuition that unbounded asynchrony is *essential* to guarantee sufficiently poor linearizability properties for counting networks whose size grows unbounded.

5.2 Upper Bound

We now prove an upper bound on the non-sequential consistency fraction under a timing condition expressing bounded asynchrony.

Theorem 5.4 (Upper Bound on Inconsistency Fraction) *Fix a uniform counting network G and a timing condition c_{min} and c_{max} such that $\frac{c_{max}}{c_{min}} < \ell$ for some integer $\ell > 1$. Then,*

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

Proof: Fix any execution $\mathcal{E}$ of G and a process p. Denote as $\mathcal{E}^p$ the sequence of tokens introduced by procees p in execution $\mathcal{E}$. We establish a technical claim:

Lemma 5.5 *Consider any arbitrary subsequence $T_1, \ldots, T_\ell$ of $\mathcal{E}^p$. Then, T_1 returns a smaller value than T_ℓ.*

Proof: Denote as $[t_{in}, t_{out}]$ and $[t'_{in}, t'_{out}]$ the time intervals during which tokens T_1 and T_ℓ traverse the network, respectively. Recall that the time for each token to traverse the network is at least $\mathsf{d}(G) \cdot c^P_{min}$. Since there are $\ell - 2$ tokens in between T_1 and T_ℓ, it follows that

$$\begin{aligned}
& t'_{in} - t_{out} \\
\geq \quad & (\ell - 2) \cdot \mathsf{d}(G) \cdot c^P_{min} \\
> \quad & \left(\frac{c_{max}}{c_{min}} - 2\right) \cdot \mathsf{d}(G) \cdot c^P_{min} \quad \text{(by assumption)} \\
\geq \quad & \left(\frac{c_{max}}{c^P_{min}} - 2\right) \cdot \mathsf{d}(G) \cdot c^P_{min} \\
= \quad & \mathsf{d}(G) \cdot (c_{max} - 2\, c^P_{min})\,.
\end{aligned}$$

Hence, Corollary 4.3 implies that T_1 returns a smaller value than T_ℓ. ∎

Lemma 5.5 implies that in the sequence $\mathcal{E}^P$, any token T_i appearing $\ell - 1$ positions before token T_j will return a smaller value. So, for each process P, remove from $\mathcal{E}$ all tokens whose order in $\mathcal{E}^P$ is different than 1 modulo $(\ell - 1)$. Clearly, the fraction of removed tokens is at most $\frac{\ell - 2}{\ell - 1}$, and the result is a sequentially consistent timed execution. The claim now follows from Lemma 5.1. ■

5.3 Lower Bounds

We now prove a lower bound on the non-linearizability and non-sequential consistency fractions of any counting network with a certain topological structure. We first need some definitions, all of which refer to a fixed balancing network G with fan-out w_{out}; we shall sometimes omit explicit mention of G.

For an output wire j of a balancer in the network G, define the ***valency*** of the wire j, denoted as $\mathsf{Val}(j)$, as the set of sink nodes reachable from j. The ***valency*** of a balancer B, denoted as $\mathsf{Val}(B)$, is the union of the valencies of its output wires; so, $\mathsf{Val}(B)$ is the set of sink nodes reachable from some output wire of the balancer B.

Assume now that G is a counting network. Recall that for any particular layer ℓ of balancers, every sink node is reachable from some balancer in the layer; so, $\bigcup_{B \in \ell} \mathsf{Val}(B) = \{1, 2, \ldots, w_{out}\}$. In addition, recall that every sink node must be reachable from each balancer in layer 1; so, for a balancer B in layer 1, $\mathsf{Val}(B) = \{1, 2, \ldots, w_{out}\}$. More generally, a balancer B such that $\mathsf{Val}(B) = \{1, 2, \ldots, w_{out}\}$ will be called ***complete***.

A balancer B is ***univalent*** if for each pair of output wires j and k of B, $1 \leq j, k \leq f_{out}$, $\mathsf{Val}(j) \cap \mathsf{Val}(k) = \emptyset$. Intuitively, B is univalent if each sink node in the set $\mathsf{Val}(B)$ unambiguously determines an output wire of the balancer; so, the eventual "sink decisions" for each of its output wires are completely separated. A layer ℓ is ***univalent*** if each balancer in ℓ is univalent.

Consider now sets of integers V_1 and V_2; say that V_1 ***precedes*** V_2, denoted as $\mathsf{V}_1 \prec \mathsf{V}_2$, if every integer in V_1 is less than any integer in V_2. Say that the balancer B (with fan-out f_{out}) is ***totally ordering*** if the set $\{\mathsf{Val}(1), \ldots, \mathsf{Val}(f_{out})\}$ is totally ordered with respect to $\prec$; that is, for each pair of output wires j and k of B, either $\mathsf{Val}(y_j) \prec \mathsf{Val}(y_k)$ or $\mathsf{Val}(y_k) \prec \mathsf{Val}(y_j)$. Intuitively, for a totally ordering balancer B, the eventual "sink decisions" for each of its output wires not only do not intersect each other, but they are also totally ordered. Clearly, any totally ordering balancer is univalent, but not vice versa. A layer is ***totally ordering*** if each of its balancers is totally ordering.

The ***split depth*** of a balancing network G, denoted as $\mathsf{sd}(G)$, is the least integer ℓ, $1 \leq \ell \leq \mathsf{d}(G)$, such that the layer ℓ of G is totally ordering; intuitively, the split depth of G measures

how far into G a token needs to get before the eventual "sink decisions" about which sink node it will exit from become disjoint and totally ordered. A totally ordering layer at split depth will be called a ***split layer***. For a network G such that $\mathsf{sd}(G) < \mathsf{d}(G)$, the ***split network*** of G, denoted as $\mathsf{SP}(G)$, is the subnetwork of G consisting of layers $\mathsf{sd}(G)+1, \ldots, \mathsf{d}(G)$.

A layer ℓ is ***complete*** if each of its balancers is complete. The network G is ***complete*** if the layer $\mathsf{sd}(G)$ is complete. A layer ℓ is ***uniformly splittable*** if for each balancer B in ℓ, for any pair of output wires j and k of B, $|\mathsf{Val}(j)| = |\mathsf{Val}(k)|$. The network G is ***uniformly splittable*** if the layer $\mathsf{sd}(G)$, is uniformly splittable.

From now on, fix a complete and uniformly splittable network G made up of $(2,2)$-balancers. Fix any balancer B in the layer $\mathsf{sd}(G)$ with output wires 1 and 2, Since G is complete, the layer $\mathsf{sd}(G)$ is complete; this implies that $\mathsf{Val}(B) = \{1, \ldots, w_{out}\}$. Since G is uniformly splittable, $\mathsf{sd}(G)$ is uniformly splittable; this implies that $|\mathsf{Val}(1)| = |\mathsf{Val}(2)|$. By definition of split depth, we have that either $\mathsf{Val}(1) \prec \mathsf{Val}(2)$ or $\mathsf{Val}(2) \prec \mathsf{Val}(1)$; assume, without loss of generality, that $\mathsf{Val}(1) \prec \mathsf{Val}(2)$. It follows that $\mathsf{Val}(1) = \left\{1, \ldots, \frac{w_{out}}{2}\right\}$ and $\mathsf{Val}(2) = \left\{\frac{w_{out}}{2}+1, \ldots, w_{out}\right\}$. This implies that the split network $\mathsf{S}(G)$ can be partitioned into two subnetworks $\mathsf{S}_1(G)$ and $\mathsf{S}_2(G)$ with output wires $1, \ldots, \frac{w_{out}}{2}$ and $\frac{w_{out}}{2}+1, \ldots, w_{out}$, respectively.

We continue to prove that the original counting networks from [AHS94] are complete and uniformly splittable; moreover, we shall calculate their split depths. These properties will be needed later, when we use the bitonic and periodic counting networks as particular examples on which to apply our general result providing lower bounds on inconsistency fractions (Theorem 5.11). We start with the bitonic counting network.

Proposition 5.6 *The bitonic counting network $B(w)$ of fan w is complete and uniformly splittable with $\mathsf{sd}(B(w)) = \frac{\lg^2 w - \lg w + 2}{2}$.*

The proof of Proposition 5.6 will refer to the construction of the bitonic counting network $B(w)$ depicted in Figure 3.

Proof: Fix layer ℓ to be the first layer in the merging network $M(w)$; so, ℓ consists of a column of balancers. Each output wire of such a balancer is connected either to $M_1(\frac{w}{2})$ or to $M_2(\frac{w}{2})$; since $M_1(\frac{w}{2})$ and $M_2(\frac{w}{2})$ are disjoint, this layer is totally ordering. We prove a simple claim:

Lemma 5.7 *No layer $\ell' < \ell$ of the network $B(w)$ is totally ordering,*

Proof: Consider any layer $\ell' < \ell$. Recall that for every output wire of $B(w)$, there is a path π of length $\mathsf{d}(G)$ that includes a balancer from every layer; let B and B' be the balancers on π from layers ℓ and ℓ', respectively. Denote as $1'$ and $2'$ the two output wires of B'. Since ℓ is a complete layer, $\mathsf{Val}(B) = \{1, \ldots, w_{out}\}$. Since $\ell' < \ell$, either $\mathsf{Val}(B) \subseteq \mathsf{Val}(1')$ or $\mathsf{Val}(B) \subseteq \mathsf{Val}(2')$. It follows that either $\mathsf{Val}(1') = \{1, \ldots, w_{out}\}$ or $\mathsf{Val}(2') = \{1, \ldots, w_{out}\}$. Hence, balancer B' is not totally ordering, which implies that ℓ' is not a totally ordering layer. ■

Lemma 5.7 implies that $\mathsf{sd}(B(w)) = \mathsf{d}(B(w)) - \mathsf{d}(M(w)) + 1 = \frac{\lg^2 w - \lg w + 2}{2}$. Fix any balancer B in the layer $\mathsf{sd}(B(w))$ with output wires 1 and 2. Recall that there is a path from each input wire to each output wire of the merging network $M(w)$. Hence, the construction of the merging network implies that $\mathsf{Val}(1) = \left\{1, \ldots, \frac{w_{out}}{2}\right\}$ and $\mathsf{Val}(2) = \left\{\frac{w_{out}}{2} + 1, \ldots, w_{out}\right\}$. So, $|\mathsf{Val}(1)| = |\mathsf{Val}(2)|$, and the network $B(w)$ is complete and uniformly splittable, as needed. ■

We now consider the periodic counting network.

Proposition 5.8 *The periodic counting network $P(w)$ of fan w is complete and uniformly splittable with $\mathsf{sd}(P(w)) = \lg^2 w - \lg w + 1$.*

The proof of Proposition 5.8 will refer to the construction of the periodic counting network $P(w)$ depicted in Figure 6.

Proof: Fix layer ℓ to be the first layer in the latest block network $L(w)$ in $P(w)$; so, ℓ is the top-bottom network $TB(w)$, and it consists of a column of balancers. Each output wire of such a balancer is connected either to $L_1(\frac{w}{2})$ or to $\widehat{L}_2(\frac{w}{2})$; since $L_1(\frac{w}{2})$ and $\widehat{L}_2(\frac{w}{2})$ are disjoint, this layer is totally ordering. No earlier layer of the network $P(w)$ can be totally ordering, as each balancer in layer ℓ is complete. It follows that $\mathsf{sd}(P(w)) = (\lg w - 1) \cdot \mathsf{d}(L(w)) + 1 = \lg^2 w - \lg w + 1$.

Fix any balancer B in the layer $\mathsf{sd}(P(w))$ with output wires 1 and 2. Recall that there is a path from each input wire to each output wire of the block network. Hence, the (second) construction of the block network implies that $\mathsf{Val}(1) = \left\{1, \ldots, \frac{w_{out}}{2}\right\}$ and $\mathsf{Val}(2) = \left\{\frac{w_{out}}{2} + 1, \ldots, w_{out}\right\}$. So, $|\mathsf{Val}(1)| = |\mathsf{Val}(2)|$, and the network $P(w)$ is complete and uniformly splittable, as needed. ■

We now provide an inductive definition of a (finite) sequence of networks $\mathsf{S}^{(0)}(G), \mathsf{S}^{(1)}(G), \ldots$, which we call the ***split sequence*** for G, as follows.

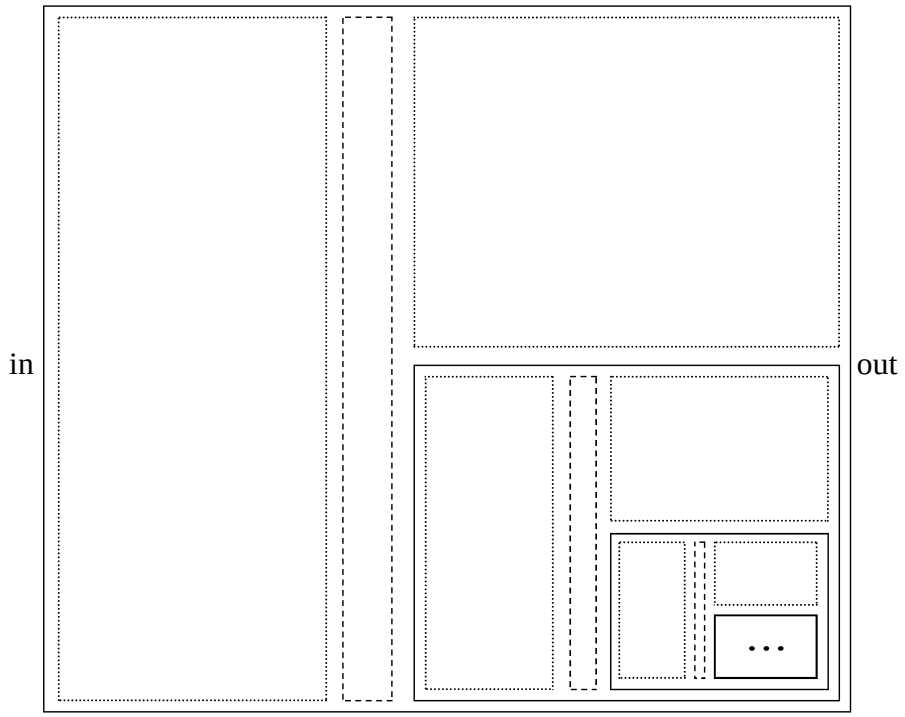

Figure 7: A continuously complete and continuously uniformly splittable network G made up of $(2,2)$-balancers in diagrammatic form. Each solid box represents a network in the split sequence for G. Each box denoted with larger dots represents a split layer. Each remaining component of the network is denoted with smaller dots. (Wires have been omitted.)

- For the basis case, take $\mathsf{S}^{(0)}(G) = G$.
- Assume inductively that we have defined the network $\mathsf{S}^{(\ell-1)}(G)$ for some integer $\ell \geq 1$.

  For the induction step, if $\mathsf{sd}(\mathsf{S}^{(\ell-1)}(G)) = \mathsf{d}(\mathsf{S}^{(\ell-1)}(G))$, then terminate; else, take $\mathsf{S}^{(\ell)}(G)$ to be $\mathsf{SP}_2(\mathsf{S}^{(\ell-1)}(G))$ (the bottom subnetwork of the split network of $\mathsf{S}^{(\ell-1)}(G)$).

Roughly speaking, the construction starts with the network G. Subsequently, each network in the sequence is obtained by "chopping off" the preceding network at its splitting depth (if possible). The ***split number*** of G, denoted as $\mathsf{sp}(G)$, is the length of the splitting sequence for G. Clearly, $\mathsf{sp}(G) \leq \mathsf{d}(G)$. The complete (resp., uniformly splittable) network G is ***continuously complete*** (resp., ***uniformly splittable***) if each network but the last in the split sequence for G is complete (resp., uniformly splittable). Figure 7 illustrates the topological structure of a continuously complete and continuously uniformly splittable network made up of $(2,2)$-balancers.

We continue to prove that the original counting networks from [AHS94] are both continuously complete and continuously uniformly splittable; moreover, we shall calculate their split numbers. We start with the bitonic counting network.

Proposition 5.9 *The bitonic counting network $B(w)$ of fan w is continuously complete and continuously uniformly splittable with* $\mathsf{sp}(B(w)) = \lg w$.

The proof of Proposition 5.9 will refer to the construction of the bitonic counting network $B(w)$ depicted in Figure 3.

Proof: By Proposition 5.6, $\mathsf{S}^{(0)}(B(w)) = B(w)$ is complete and unformly splittable. By the definition of the split sequence and the construction of the bitonic counting network, $\mathsf{S}^{(1)}(B(w))$ is the merging network $M_2\left(\frac{w}{2}\right)$. We first prove that the merging network is complete and uniformly splittable.

Fix any layer B in layer 1 of the merging network $M(w)$ with output wires 1 and 2. Each of these output wires is connected either to $M_1(\frac{w}{2})$ or to $M_2(\frac{w}{2})$; since $M_1(\frac{w}{2})$ and $M_2(\frac{w}{2})$ are disjoint, this layer is totally ordering. Recall that there is a path from each input wire to each output wire of the merging network. Hence, the construction of the merging network implies that $\mathsf{Val}(1) = \left\{1, \ldots, \frac{w_{out}}{2}\right\}$ and $\mathsf{Val}(2) = \left\{\frac{w_{out}}{2} + 1, \ldots, w_{out}\right\}$. So, $|\mathsf{Val}(1)| = |\mathsf{Val}(2)|$, so that layer 1 of the merging network is complete and uniformly splittable. It follows that the network $M(w)$ is complete and uniformly splittable.

Since layer 1 of the merging network $M(w)$ is totally ordering, complete and uniformly splittable, $\mathsf{sd}(M(w)) = 1$. By the definition of the split sequence and the construction of the merging network, $\mathsf{S}^{(2)}(M(w)) = \mathsf{SP}(\mathsf{S}^{(1)}(B(w))) = \mathsf{SP}\left(M_2\left(\frac{w}{2}\right)\right) = M_2\left(\frac{w}{4}\right)$. Hence, the argument applies inductively to imply that the network $B(w)$ is continuously complete and continuously uniformly splittable.

Besides the bitonic counting network $B(w)$ used in the basis of the inductive definition of split sequence, each layer but the last of the merging network determines a split network in the sequence. Hence, the split number of $B(w)$ is $1 + (\mathsf{d}(M(w)) - 1 = \lg w$, and the proof is complete. ■

We now consider the periodic counting network.

Proposition 5.10 *The periodic counting network $P(w)$ of fan w is continuously complete and continuously uniformly splittable with* $\mathsf{sp}(P(w)) = \lg w$.

The proof of Proposition 5.10 will refer to the construction of the periodic counting network $P(w)$ depicted in Figure 6.

Proof: By Proposition 5.8, $\mathsf{S}^{(0)}(P(w)) = P(w)$ is complete and unformly splittable. By the definition of the split sequence and the construction of the periodic counting network, $\mathsf{S}^{(1)}(P(w))$ is the block network $\widehat{L}_2\left(\frac{w}{2}\right)$ that is included in the last block of the periodic counting network. We first prove that the block network is complete and uniformly splittable.

Fix any balancer B in layer 1 of the block network $\widehat{L}(w)$ with output wires 1 and 2. Each of these output wires is connected either to $\widehat{L}_1\left(\frac{w}{2}\right)$ or to $\widehat{L}_2\left(\frac{w}{2}\right)$; since $\widehat{L}_1\left(\frac{w}{2}\right)$ and $\widehat{L}_2\left(\frac{w}{2}\right)$ are disjoint, this layer is totally ordering. Recall that there is a path from each input wire to each output wire of the block network. Hence, the construction of the block network implies that $\mathsf{Val}(1) = \left\{1, \ldots, \frac{w_{out}}{2}\right\}$ and $\mathsf{Val}(2) = \left\{\frac{w_{out}}{2} + 1, \ldots, w_{out}\right\}$. So, $|\mathsf{Val}(1)| = |\mathsf{Val}(2)|$, so that layer 1 of the block network is complete and uniformly splittable. It follows that the network $\widehat{L}(w)$ is complete and uniformly splittable.

Since layer 1 of the block network $\widehat{L}(w)$ is totally ordering, complete and uniformly splittable, $\mathsf{sd}(\widehat{L}(w)) = 1$. By the definition of the split sequence and the construction of the block network, $\mathsf{S}^{(2)}(P(w)) = \mathsf{SP}\left(\mathsf{S}^{(1)}(P(w))\right) = \mathsf{SP}\left(\widehat{L}_2\left(\frac{w}{2}\right)\right) = \widehat{L}_2\left(\frac{w}{4}\right)$. Hence, the argument applies inductively to imply that the network $\widehat{L}(w)$ is continuously complete and continusously uniformly splittable.

Besides the periodic counting network $P(w)$ used in the basis of the inductive definition of split sequence, each layer but the last of the block network determines a split network in the sequence. Hence, the split number of $P(w)$ is $1 + (\mathsf{d}(\widehat{L}(w)) - 1 = \lg w$, and the proof is complete. ■

We are now ready to prove our lower bounds on inconsistency fractions:

Theorem 5.11 (Lower Bounds on Inconsistency Fractions) *Fix a uniform, continuously complete and continuously uniformly splittable counting network $G(w)$ with fan w, and a timing condition c_{min} and c_{max} such that*

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

for some integer ℓ such that $1 \leq \ell \leq \mathsf{sp}(G(w))$. Then,

$$\mathbf{F}_{nl}(G(w)) \quad \geq \quad 1 - \frac{1}{2 - \left(\frac{1}{2}\right)^{\ell}}\,.$$

and

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

The proof of Theorem 5.11 extends the proof of Proposition 5.3 to apply to counting networks with a topological structure generalizing that of the bitonic counting network. The proof will exploit the general structure of a continuously complete and continuously uniformly splittable counting network G, as illustrated in Figure 7.

Proof: We construct a (finite) timed execution $R_{\mathcal{E}}$ of the network $G(w)$ with three *waves* of tokens; the tokens in each wave enter the network simultaneously and through a different input wire, and they proceed in lock step through the network. Each of the first and third waves contains $\frac{w}{2}+\ldots+\frac{w}{2^{\ell}} = w\cdot\left(1-\left(\frac{1}{2}\right)^{\ell}\right)$ tokens. The second wave contains $\frac{w}{2^{\ell}}$ tokens.

- The first wave enters through the input wires $1,\ldots,w\cdot\left(1-\left(\frac{1}{2}\right)^{\ell}\right)$, with one token per wire. The speed of this wave in the network is the slowest possible – one wire per time c_{max}.

  Since $G(w)$ is a counting network, these tokens must proceed to the output wires $1,\ldots,w\cdot\left(1-\left(\frac{1}{2}\right)^{\ell}\right)$. Since the network $G(w)$ is continuously complete and continuously uniformly splittable, $\frac{w}{2^{i}}$ of them will be exiting through the network $\mathsf{SP}_1\left(\mathsf{S}^{(i-1)}(G(w))\right)$ (the top subnetwork of the split network of $\mathsf{S}^{(i-1)}(G(w))$), where $1\leq i\leq \ell$.

- Immediately following the first wave (and with no relative delay) is the second wave, which enters through the input wires $1,\ldots,\frac{w}{2^{\ell}}$, with one token per wire. Assume that the tokens in the second wave are introduced by processes $p_1,\ldots,p_{\frac{w}{2^{\ell}}}$.

  Since $G(w)$ is a counting network, these tokens must proceed to the output wires $w-\frac{w}{2^{\ell}}+1 = w\cdot\left(1-\frac{1}{2^{\ell}}\right)+1,\ldots,w$. Since the network $G(w)$ is continuously complete and continuously uniformly splittable, these tokens must exit through the network $\mathsf{S}^{(\ell)}(G(w)) = \mathsf{SP}_2\left(\mathsf{S}^{(\ell-1)}(G(w))\right)$ (the bottom subnetwork of the split network of $\mathsf{S}^{(\ell-1)}(G(w))$).

  The speed of this wave in the network $\mathsf{SP}_2\left(\mathsf{S}^{(\ell-1)}(G(w))\right)$ is the fastest possible – one wire per time c_{min}. Before reaching this network, the speed of this wave is the slowest possible– one wire per time c_{max}. Since the network $G(w)$ is uniform, this implies that the first and second waves will be simultaneously entering the networks $\mathsf{SP}_1\left(\mathsf{S}^{(\ell-1)}(G(w))\right)$ and $\mathsf{SP}_2\left(\mathsf{S}^{(\ell-1)}(G(w))\right)$, respectively.

- As soon as the second wave exits the network G, the third wave enters the network through the input wires $1, \ldots, w \cdot \left(1 - \left(\frac{1}{2}\right)^{\ell}\right)$, with one token per wire. The speed of this wave in the network is the fastest possible – one wire per time c_{min}. Assume that there are $\frac{w}{2^{\ell}}$ tokens in this wave that are introduced by processes $p_1, \ldots, p_{\frac{w}{2^{\ell}}}$.

  Since $G(w)$ is a counting network, these tokens must proceed to the output wires $1, \ldots, w \cdot \left(1 - \left(\frac{1}{2}\right)^{\ell}\right)$. Since the network $G(w)$ is continuously complete and continuously uniformly splittable, $\frac{w}{2^{\ell'}}$ of them will be exiting through the network $\mathsf{SP}_1\left(\mathsf{S}^{(\ell'-1)}(G(w))\right)$, where $1 \leq \ell' \leq \ell$.

Clearly, the values returned to the tokens in the second wave are $w \cdot \left(1 - \frac{1}{2^{\ell}}\right), \ldots, w - 1$. We pursue a timing analysis to determine the values returned to tokens in the third wave. Denote as t_1, t_2 and t_3 the times for the first, second and third waves to reach the counters of the network; these times are taken from the point where tokens of the first wave are entering the network $\mathsf{SP}_1\left(\mathsf{S}^{(\ell-1)}(G(w))\right)$ Recall that the second wave is simultaneously entering the network $\mathsf{S}^{(\ell)}(G(w)) = \mathsf{SP}_1\left(\mathsf{S}^{(\ell-1)}(G(w))\right)$. Since the network $G(w)$ is uniform, it follows that $\mathsf{d}\left(\mathsf{SP}_1\left(\mathsf{S}^{(\ell-1)}(G(w))\right)\right) = \mathsf{d}\left(\mathsf{SP}_2\left(\mathsf{S}^{(\ell-1)}(G(w))\right)\right) = \mathsf{d}\left(\mathsf{S}^{(\ell)}(G(w))\right)$. Hence, $t_1 = \mathsf{d}\left(\mathsf{S}^{(\ell)}(G(w))\right) \cdot c_{max}$ and $t_2 = \mathsf{d}\left(\mathsf{S}^{(\ell)}(G(w))\right) \cdot c_{min}$. On the other hand, $t_3 = t_2 + \mathsf{d}\left(G(w)\right) \cdot c_{min}$. It follows that

$$\begin{array}{rcll} & t_3 & & \\ = & \left(\mathsf{d}\left(\mathsf{S}^{(\ell)}(G(w))\right) + \mathsf{d}\left(G(w)\right)\right) \cdot c_{min} & & \\ < & \mathsf{d}\left(\mathsf{S}^{(\ell)}(G(w))\right) \cdot c_{max} & & \text{(by assumption)} \\ = & t_1\,. & & \end{array}$$

This implies that the third wave bypasses the first and obtains values $0, \ldots, w \cdot \left(1 - \left(\frac{1}{2}\right)^{\ell}\right) - 1$.

- Note that each value returned to a token in the third wave is smaller than any value returned to a token in the second wave. So, there are $w \cdot \left(1 - \left(\frac{1}{2}\right)^{\ell}\right)$ non-linearizable tokens in the execution $\mathcal{E}$ with $w + w \cdot \left(1 - \left(\frac{1}{2}\right)^{\ell}\right)$ tokens. It follows that

$$\begin{array}{rcl} \mathbf{F}_{nl}(G(w)) & \geq & \dfrac{1 - \left(\frac{1}{2}\right)^{\ell}}{2 - \left(\frac{1}{2}\right)^{\ell}} \\ & = & 1 - \dfrac{1}{2 - \left(\frac{1}{2}\right)^{\ell}}\,. \end{array}$$

- Each value returned to a token introduced by some process p_l, $1 \leq l \leq \frac{w}{2^\ell}$, in the third wave is smaller than the value returned to the token by the same process in the second wave. So, there are $w \cdot \frac{1}{2^\ell}$ non-sequentially consistent tokens in the execution $\mathcal{E}$ with $w + w \cdot \left(1 - \left(\frac{1}{2}\right)^\ell\right)$. tokens. It follows that

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

The proof is now complete. ∎

Theorem 5.11 establishes a collection of lower bounds on the non-linearizability and non-sequential consistency fractions, together with a corresponding collection of timing conditions on c_{min} and c_{max}. Each timing condition and the corresponding lower bounds on the inconsistency fractions are determined by the integer ℓ, where $1 \leq \ell \leq \mathsf{sp}(G(w))$.

- Each timing condition takes the form of a lower bound on the ratio $\frac{c_{max}}{c_{min}}$. The lower bound depends on ℓ since it involves $\mathsf{d}(\mathsf{S}^{(\ell)}(G(w)))$ As ℓ increases, $\mathsf{d}(\mathsf{S}^{(\ell)}(G(w)))$ decreases, and, therefore, the assumed lower bound increases as well. Thus, the lower bounds on inconsistency fractions corresponding to larger (resp., smaller) values of l apply to distributed systems with stronger asynchrony (resp., weaker) asynchrony.

- The established lower bound on $\mathbf{F}_{nl}(G(w))$ increases with ℓ; it approaches $\frac{1}{2}$ as ℓ increases. To the contrary, the established lower bound on $\mathbf{F}_{nsc}(G(w))$ decreases with ℓ; it approaches 0 as ℓ increases. Propositions 5.9 and 5.10 imply that the largest value taken on by ℓ (namely, $\mathsf{sp}(G(w))$ becomes larger when the counting network $G(w)$ becomes larger in the case where the counting network is the bitonic counting network or the periodic counting network, respectively; we conjecture that the same holds for other known families of counting networks (e.g., the counting network in [BM98]). This observation appears to suggest 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; however, the suggsetion would only be relevant if the lower bounds on $\mathbf{F}_{nl}(G(w))$ and $\mathbf{F}_{nsc}(G(w))$ were shown to be *tight*.

  Remarkably, the established lower bounds on the non-linearizability and non-sequential consistency fractions coincide for the smallest value $\ell = 1$ to the value $\frac{1}{3}$. For the particular

case of the bitonic counting network, these lower bounds have been established by Lynch *et al* [LSST99] (see Proposition 5.2 in this paper) for linearizability, and in this paper (Proposition 5.3) for sequential consistency, respectively. Taking $\ell = 1$ in Theorem 5.11, and using Proposition 5.8, implies that identical lower bounds on inconsistency fractions hold for the periodic counting network as well.

In conclusion, the established lower bounds on the non-linearizability and non-sequential consistency fractions tend to diverge in systems with strong asynchrony but converge in systems with weak asynchrony.

We conclude this section with some particular examples of applying Theorem 5.11. We shall examine the case where $\ell = \mathsf{sp}(G(w))$ in Theorem 5.11, for the particular cases of the bitonic and periodic counting networks, which are both uniform, continuously complete and continuously uniformly splittable (by Propositions 5.6 and 5.8, respectively).

Recall that for the bitonic counting network $B(w)$, $\mathsf{d}(B(w)) = \frac{\lg w(\lg w + 1)}{2}$, $\mathsf{sp}(B(w)) = \lg w$ (by Proposition 5.9) and $\mathsf{d}\left(\mathsf{S}^{(\mathsf{sp}(B(w)))}(B(w))\right) = 1$. We use Theorem 5.11 with $\ell = \lg w$; then, $1 - \frac{1}{2 - \left(\frac{1}{2}\right)^{\ell}}$ becomes $\frac{w-1}{2w-1}$ and $\frac{\left(\frac{1}{2}\right)^{\ell}}{2 - \left(\frac{1}{2}\right)^{\ell}}$ becomes $\frac{1}{2w-1}$. Hence, we obtain:

Corollary 5.12 *Consider the bitonic counting network $B(w)$ with fan w, under a timing condition c_{min} and c_{max} such that $\frac{c_{max}}{c_{min}} > 1 + \frac{\lg w(\lg w + 1)}{2}$. Then,*

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

and

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

Recall that for the periodic counting network $P(w)$, $\mathsf{d}(P(w)) = \lg^2 w$, $\mathsf{sp}(P(w)) = \lg w$ (by Proposition 5.10) and $\mathsf{d}\left(\mathsf{S}^{(\mathsf{sp}(P(w)))}(P(w))\right) = 1$. We use Theorem 5.11 with $\ell = \lg w$; then, $1 - \frac{1}{2 - \left(\frac{1}{2}\right)^{\ell}}$ becomes $\frac{w-1}{2w-1}$ and $\frac{\left(\frac{1}{2}\right)^{\ell}}{2 - \left(\frac{1}{2}\right)^{\ell}}$ becomes $\frac{1}{2w-1}$. Hence, we obtain:

Corollary 5.13 *Consider the periodic counting network $P(w)$ with fan w, under a timing condition c_{min} and c_{max} such that $\frac{c_{max}}{c_{min}} > 1 + \lg^2 w$. Then,*

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

and

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

6 Open Problems

We conclude this article with a rich collection of problems that remain open:

1. Identify further timing conditions that cannot distinguish sequential consistency from linearizability in uniform counting networks.
2. Identify timing conditions that cannot distinguish sequential consistency from linearizability in *general* (not necessarily uniform) counting networks.
3. Identify timing conditions that distinguish sequential consistency from linearizability in *general* (not necessarily uniform) counting networks.
4. Establish tightness (or show an improvement) for the upper bound on the absolute non-sequential consistency fraction in Theorem 5.4. Establish more upper and lower bounds on absolute inconsistency fractions.
5. Establish tightness (or show improvements) for the lower bounds on non-linearizability and non-sequential consistency fractions in Theorem 5.11.

Acknowledgements.

We would like to thank the anonymous *Distributed Computing* referees for their many valuable comments.

References

[AA95] E. Aharonson and H. Attiya, "Counting Networks with Arbitrary Fan-out," *Distributed Computing,* Vol. 8, No. 4, 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.

[AM94] H. Attiya and M. Mavronicolas, "Efficiency of Semi-Synchronous versus Asynchronous Networks," *Mathematical Systems Theory,* Vol. 27, No. 6, pp. 547–571, November/December 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.

[BMT02] H. Brit, S. Moran and G. Taubenfeld, "Public Data Structures: Counters as a Special Case," *Theoretical Computer Science,* Vol. 289, No. 1, pp. 401–423, November 2002.

[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.

[BH02] C. Busch and M. Herlihy, "Sorting and Counting Networks of Arbitrary Width and Small Depth," *Theory of Computing Systems,* Vol. 35, No. 2, pp. 99–128, January 2002.

[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.

[BMS05] C. Busch, M. Mavronicolas and P. Spirakis, "The Cost of Concurrent, Low-Contention Read&Modify&Write," *Theoretical Computer Science,* Vol. 333, No. 4, pp. 373–400, March 2005.

[EM99] M. Eleftheriou and M. Mavronicolas, "Linearizability in the Presence of Drifting Clocks and Under Different Delay Assumptions," *Proceedings of the 13th International Symposium on Distributed Computing,* P. Jayanti ed. pp. 327–341, Vol. 1693, Lecture Notes in Computer Science, Springer-Verlag, September 1999.

[FH04] P. Fatourou and M. Herlihy, "Read-Modify-Write Networks," *Distributed Computing,* Vol. 17, No. 1, pp. 33–46, February 2004.

[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.

[FG91] E. Freudenthal and A. Gottlieb, "Process Coordination with Fetch-and-Increment," *Proceedings of the 4th ACM International Conference on Architectural Support for Programming Languages and Operating Systems,* pp. 260–268, April 1991.

[GVW89] J. R. Goodman, M. K. Vernon and P. J. Woest, "Efficient Synchronization Primitives for Large-Scale Cache-Coherent Multiprocessors," *Proceedings of the 3rd ACM International Conference on Architectural Support for Programming Languages and Operating Systems,* pp. 64–75, April 1989.

[HKM93] N. Hardavellas, D. Karakos and M. Mavronicolas, "Notes on Sorting and Counting Networks," *Proceedings of the 7th International Workshop on Distributed Algorithms,* A. Schiper ed., pp. 234–248, Vol. 725, Lecture Notes in Computer Science, Springer-Verlag, September 1993.

[HSW96] M. Herlihy, N. Shavit and O. Waarts, "Linearizable Counting Networks," *Distributed Computing,* Vol. 9, No. 4, pp. 193–203, February 1996.

[HT06a] M. Herlihy and S. Tirthapura, "Self-stabilizing Smoothing and Balancing Networks," *Distributed Computing,* Vol. 18, No. 5, pp. 345–357, April 2006.

[HT06] M. Herlihy and S. Tirthapura, "Randomized Smoothing Networks," *Journal of Parallel and Distributed Computing,* Vol. 66, No. 5, pp. 626–632, May 2006.

[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.

[LSST99] N. Lynch, N. Shavit, A. Shvartsman and D. Touitou, "Timing Conditions for Linearizability in Uniform Counting Networks," *Theoretical Computer Science,* Vol. 220, No. 1, pp. 67–92, June 1999.

[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.

[MR99] M. Mavronicolas and D. Roth, "Linearizable Read/Write Objects," *Theoretical Computer Science,* Vol. 220, No. 1, pp. 267–319, June 1999.

[ML89] J. M. Mellor-Crummey and T. J. LeBlanc, "A Software Instruction Counter," *Proceedings of the 3rd ACM International Conference on Architectural Support for Programming Languages and Operating Systems,* pp. 78–86, April 1989.

[MS91] J. M. Mellor-Crummey and M. L. Scott, "Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors," *ACM Transactions on Computer Systems,* Vol. 9, No. 1, pp. 21–65, February 1991.

[MT97] S. Moran and G. Taubenfeld, "A Lower Bound on Wait-Free Counting," *Journal of Algorithms,* Vol. 24, No. 1, pp. 1–19, July 1997.

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

[SZ96] N. Shavit and A. Zemach, "Diffracting Trees," *ACM Transactions on Computer Systems,* Vol. 14, No. 4, pp. 385–428, November 1996.

[T06] G. Taubenfeld, *Synchronization Algorithms and Concurrent Programming,* Pearson Education/Prentice-Hall, 2006.

[T05] S. Tirthapura, "Adaptive Counting Networks," *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems,* pp. 241–250, June 2005.

A Bibliographical Notes

Marios Mavronicolas. He is a Professor of Computer Science at University of Cyprus. His research interests span the Theory of Algorithms and Complexity, with focus on Game Theory, Distributed and Parallel Computing, Networking and the Internet, where he has published widely in leading journals and conferences. He is on the Editorial Board of *Theoretical Computer Science, Journal of Interconnection Networks* and *Networks*. He previously held faculty positions at the University of Connecticut and the University of Crete. He holds a PhD in Computer Science from Harvard University.

Michael Merritt. He is Director of the Network Design and Analysis Research Department at AT&T Labs–Research. This department is responsible for fundamental and applied research in Efficient Algorithms and their applications to Communication Networks. Michael is a recognized expert in Distributed Computing and Computer Security. He is an Area Editor for *Distributed Computing* and the *Journal of the ACM*. He has taught at Georgia Tech, MIT, Stevens Institute of Technology and Columbia University.

Gadi Taubenfeld. He is an Associate Professor of Computer Science at the Interdisciplinary Center in Herzliya, Israel. He is an established authority in the area of Concurrent and Distributed Computing, where he has published widely in leading journals and conferences. He was the Head of the Computer Science Division at Israel's Open University; member of technical staff at AT&T Bell Laboratories; consultant to AT&T Labs-Research; and a research scientist and lecturer at Yale University. He holds a PhD in Computer Science from the Technion - Israel Institue of Technology.