The Cost of Concurrent, Low-Contention Read-Modify-Write*

COSTAS BUSCH
Rensselaer Polytechnic Institute, USA

MARIOS MAVRONICOLAS
University of Cyprus, Cyprus

PAUL SPIRAKIS
University of Patras, Greece

Abstract

In this work, we embark on a study of the possibility (or impossibility), and the corresponding costs, of devising concurrent, low-contention implementations of atomic Read-Modify-Write operations (abbreviated as RMW), in a distributed system. We consider a natural class of RMW operations which give rise to a certain class of algebraic groups that we introduce here and call *monotone groups*. Our chief combinatorial instrument is a *Monotone Linearizability Lemma*, which establishes inherent ordering constraints of linearizability for a certain class of executions of *any* distributed system that implements a monotone RMW operation.

The end results of our study specifically apply to implementations of (monotone) RMW operations that are based on *switching networks*, a recently introduced class of concurrent, low-contention data structures that generalize *counting networks*. These results are negative and they are shown through a modular use of the Monotone Linearizability Lemma. In particular, we derive the *first* lower bounds on *size* (the number of *switches* in the network) and *latency* (the maximum number of switches traversed) for any (non-trivial) switching network implementing a monotone RMW operation:

- If the network is made up of switches with *finite* state and it incurs low contention, then it must contain an infinite number of switches, even if *concurrency* (the maximum number of concurrent processes) is restricted to remain bounded.

*This work has been partially supported by the IST Program of the European Union under contract numbers IST-1999-14186 (ALCOM-FT) and IST-2001-33116 (FLAGS), by funds from the Joint Program of Scientific and Technological Collaboration between Greece and Cyprus, by the Greek General Secretariat for Research and Technology, and by research funds from Rensselaer Polytechnic Institute and University of Cyprus.

- If the network is made up of switches with *infinite* state and it incurs low contention, then it must still contain an infinite number of switches if we now allow concurrency to grow unbounded.
- Any switching network induces executions with latency at least $\lceil \frac{n-1}{c-1} \rceil$, where n is the number of concurrent processes and c is the maximum number of processes that simultaneously access a switch.

A major significance of the above lower bounds (of infinity) is that they formally explain the observed inability of researchers over the last decade to extend counting networks, while retaining them finite and low-contention, in order to perform tasks more complex than just incrementing a counter by one.

Keywords

concurrency, read-modify-write operations, linearizability, switching networks

1 Introduction

Motivation, Framework and Outline. A Read-Modify-Write shared variable [8, 13], henceforth abbreviated as RMW, is an abstract variable type that allows reading its old value, and determining (via some specific *operator*) and writing a new value back to it in a single, *atomic* (indivisible) RMW operation (cf. [15, Example 9.4.2]). A RMW operation is a strong synchronization primitive that allows for the design of efficient and transparent algorithms in the asynchronous shared memory model of distributed computation; see, e.g., the folklore algorithm for mutual exclusion described in [3, Section 4.3.2], or the scalable *ordered multicast* protocol of Herlihy *et al.* [11] that is based on a modular use of the distributed Swap operation, a special case of RMW. Due to their fundamental importance as synchronization primitives, it is most desirable to devise suitable distributed data structures for the construction of concurrent, low-contention implementations of RMW variables. Intuitively, the *contention* of an implementation measures the extent to which concurrent *processes* access the same memory location simultaneously; it has been argued quite convincingly that contention is a critical factor for the overall efficiency of shared memory algorithms (cf. [6]). The central question motivating this work is the possibility (or impossibility), and the corresponding incurred complexities, for concurrent, low-contention implementations of RMW shared variables.

We focus on a specific class of RMW operations whose associated operators give rise to a certain class of algebraic groups introduced and studied here, which we call *monotone groups*. A monotone group has a *total order* and a *monotone subdomain* associated with it; the latter enjoys a significant monotonicity property, which we call *monotonicity under composition*: applying the operator on an

element from the monotone subdomain results to another element in the monotone subdomain that strictly dominates the initial one with respect to the total order. For example, the Fetch&Add operation (over the set of integers) clearly falls into the context of monotone groups, since adding a positive integer to a positive integer results in a larger positive integer; here, the monotone subdomain is the set of positive integers. So also does the Fetch&Multiply operation, and so on. A *monotone* RMW operation is one that is associated with a monotone group.

An abstract concept defined in relation to monotone groups is that of *n-wise independence*. Roughly speaking, n elements of a monotone group are *n-wise independent* if it is not possible to derive the identity element of the group through successive (specifically restricted though) applications of the operator on n of the elements or their inverses. A preliminary but significant property of monotone groups that we prove is that *every* monotone group is n-wise independent, in the sense of having n-wise independent elements. As we establish, the existence of n-wise independent elements in a monotone group is largely responsible for enforcing *linearizability* [12] for certain suitable executions of a distributed system that implements the corresponding (monotone) RMW operation; recall that an execution is *linearizable* [12] if the values returned to operations in it respect the real-time ordering of the operations.

As a consequence, the main conclusion of our work is that guaranteeing the inherent linearizability for these particular executions must incur a high cost in efficiency for a certain class of concurrent, low-contention implementations of (monotone) RMW that are based on switching networks; these are concurrent, low-contention data structures that were recently introcuded [7] as a generalization of *counting networks* [2]. Roughly speaking, a *switching network* is a directed, acyclic graph made up of *switches* and *output registers*; whenever a process issues a RMW operation, it shepherds a *token* through the network, which traverses a path of switches till it is eventually returned a value (at an output register). Thus, concurrent processes are spatially dispersed in a switching network, which reduces their simultaneous crossings in front of the same memory location; this offers potential for low contention. The *size* of a switching network is the total number of switches in it; its *latency* is the maximum number of switches traversed by a token shepherding a RMW operation through the network. The *concurrency* of a switching network is the maximum number of concurrent *processes* that may shepherd a RMW operation through the network.

In order to model the low-contention property for switching networks, we introduce *register bottleneck* and *layer bottleneck*; roughly speaking, both register bottleneck and layer bottleneck measure the *minimum* number of network elements (either switches or output registers) that are accessed by processes in any infinite execution. (Layer bottleneck assumes partitioning the switches of the network into *layers* in the natural way.) Intuitively, if this minimum number is small, some network element will become a *bottleneck* (or a *"hot-spot"* in the pool of memory locations) in some infinite execution and the network incurs high con-

tention; hence, a switching network is *low-contention* if register bottleneck and layer bottleneck are sufficiently large.

Contribution. Our chief combinatorial instrument is a *Monotone Linearizability Lemma* (Proposition 1), which establishes inherent ordering constraints of linearizability for a certain class of executions of *any* distributed system that implements a monotone RMW operation. Interestingly, in these executions, the arguments of the operations performed by the concurrent processes enjoy together the group-theoretic property of n-wise independence over the associated monotone group.

The end results of our study are negative; they are shown through a modular use of the Monotone Linearizability Lemma. The corresponding general methodology we propose for showing impossibility results for any given class of distributed implementations of monotone RMW operations is to devise suitable counter-example executions which, on one hand, are inherently linearizable (courtesy of the Monotone Linearizability Lemma), while, on the other hand, they are suitably constructed so as to compromise linearizability and thus contradict the Monotone Linearizability Lemma. Applying this general methodology to implementations based on switching networks yields the *first* lower bounds on size and latency for a low-contention switching network that implements a monotone RMW operation. We obtain the following results for any switching network other than the trivial single-switch one:

- If the switching network is made up of *switches* with *finite* state and it is low-contention, then it must contain an *infinite* number of switches, even if concurrency is restricted to remain *bounded* (Theorem 1).
- If the switching network is made up of switches with *infinite* state and it is low-contention, then it must still contain an *infinite* number of switches if concurrency is now allowed to grow unbounded (Theorem 2).

We note that our two lower bounds on the size of any switching network that implements a monotone RMW operation represent a trade-off between the strength of the switches (finite or infinite state) and the concurrency of the network (bounded or unbounded). Thus, neither of them is implied by the other. Our final result deals with latency. We obtain:

- Any switching network (whether made up of switches of finite or infinite state) that implements a monotone RMW operation induces executions with latency at least $\lceil \frac{n-1}{c-1} \rceil$, where n is the number of concurrent processes participating in the execution, and c, the network's *capacity,* is the maximum number of processes that simultaneously access a switch in any execution of the network.

Our impossibility results for switching networks indicate that inherent linearizability, necessitated by our Monotone Linearizability Lemma, is the crucial

bottleneck that rules out efficiency (with respect to both size and latency) for any low-contention switching network that implements a monotone RMW operation. In fact, we believe that inherent linearizability is indeed the crucial efficiency bottleneck for *any* such class of distributed, low-contention implementations, but this remains to be seen. Finally, we remark that linearizability has so far been studied as a *required* property for a distributed system that best guarantees acceptable concurrent behavior. To the best of our knowledge, our work is the *first* to provide, through the Monotone Linearizability Lemma, a (non-trivial) instance of a distributed system where linearizability is an *inherent* property.

Related Work, Comparison and Significance. The notion of linearizability has been introduced by Herlihy and Wing [12]. Switching networks (and, in particular, *adding networks*) were recently studied in [7], as an extension to counting networks [2] that accommodates the general Fetch&Add operation (as opposed to the Fetch&Increment and Fetch&Decrement operations that were supported before by counting networks [1, 2, 18]); for more on counting networks, see, e.g., [4, 5, 10, 16, 17].

Theorems 1 and 2 settle to the negative a far generalization of a specific open question articulated in [7, Section 5] about the existence of switching networks with a *finite* number of switches that implement the (monotone) Fetch&Add operation. (Two solutions, called adding networks, with an *infinite* number of switches were presented in [7, Section 4].) Indeed, the more general problem of devising *finite* network-based data structures, as suitable extensions to counting networks, to support synchronization operations other than Fetch&Increment (which was originally supported by counting networks) was already stated in the seminal work of Aspnes *et al.* [2] that introduced counting networks; however, it has remained essentially open: progress on this problem has been so far limited to discovering that counting networks themselves can also support Fetch&Decrement (concurrently with Fetch&Increment)[1, 18]. The impossibility results established in Theorems 1 and 2 provide a mathematical explanation for the apparent lack of progress on this problem; thus, they are significant since they explain the observed inability of researchers in the last decade or so (since the original conference publication of counting networks [2]) to operationally extend counting networks, while still retaining them finite and low-contention, in order to perform tasks more complex than just incrementing a counter by one but yet as simple as adding an arbitrary value to a counter.

The structure of the proofs of Theorems 1 and 2 is inspired by that of the proof of a result of Herlihy *et al.* [10, Theorem 5.1], showing that any (non-blocking) counting network [2] (other than the trivial single-balancer one) must have an *infinite* number of balancers if all of its executions are to be linearizable. The requirement that *all* executions be linearizable allows the proof of [10, Theorem 5.1] to pick the execution of choice and force it to violate linearizability. However, a switching network for a monotone RMW operation need not guarantee

linearizability in all executions; thus, the role of the Monotone Linearizability Lemma is to contribute to the proofs of Theorems 1 and 2 executions that are necessarily linearizable. Note also that although a counting network is a special case of a switching network, the lower bound on size established in [10, Theorem 5.1] for a linearizable counting network does not immediately apply to switching networks that implement a monotone RMW operation, since the proof of [10, Theorem 5.1] relies on the behavior of counting networks; instead, the proofs of Theorems 1 and 2 require far more delicate arguments that are specific to the behavior of switching networks.

Theorem 3 is reminiscent of a recent result [7, Theorem 1] that establishes a lower bound of $\lceil \frac{n-1}{c-1} \rceil$ on latency for adding networks, where there are only two possible arguments for addition, namely a and b such that $|a| > |b| > 0$; more specifically, it is shown that each token of weight b traverses at least $\lceil \frac{n-1}{c-1} \rceil$ switches, while, if also $|b| > 1$, each token of weight a traverses at least $\lceil \frac{n-1}{c-1} \rceil$ switches. Theorem 3 significantly extends and improves [7, Theorem 1] in the following ways: First, Theorem 3 applies to switching networks that implement *any* monotone RMW operation, while [7, Theorem 1] is specific to adding networks and the Fetch&Add operation, and second, despite the enhanced generality of Theorem 3, its proof is far simpler and more natural and succinct than that of [7, Theorem 1].

2 Monotone Groups

Basic Definitions. We start by reviewing some very basic definitions from Group Theory. (See [9] for a general background in Group Theory.) A (binary) *operator* (also called *composition law*) on a set Γ is a mapping $\oplus : \Gamma \times \Gamma \to \Gamma$. A *group* $\langle \Gamma, \oplus \rangle$ is a set Γ together with an operator $\oplus$ such that: *(1) Closure Property:* for all pairs of elements $a, b \in \Gamma$, $a \oplus b \in \Gamma$, *(2) Associativity:* for all triples of elements $a, b, c \in \Gamma$, $(a \oplus b) \oplus c = a \oplus (b \oplus c)$, *(3) Identity Element:* there is an element $a \in \Gamma$, called the *identity element* of Γ, such that for each element $a \in \Gamma$, $a \oplus e = e \oplus a = a$, and *(4) Inverse Element:* for each element $a \in \Gamma$, there is an element $a^{-1} \in \Gamma$, called the *inverse* of a, such that $a \oplus a^{-1} = a^{-1} \oplus a = e$. An *Abelian group* is a group $\langle \Gamma, \oplus \rangle$ which satisfies in addition the following property: *(5) Commutativity:* for all pairs of elements $a, b \in \Gamma$, $a \oplus b = b \oplus a$.

Composite Operators. We proceed to define two composite operators by applying the operator $\oplus$ a number of times. For any integer k, define the unary operator $\bigoplus_k : \Gamma \to \Gamma$ as follows: $\bigoplus_k a = a \oplus \ldots \oplus a$ k times if $k > 0$, e if $k = 0$, and $a^{-1} \oplus a^{-1} \oplus \ldots \oplus a^{-1}$ $-k$ times if $k < 0$. Call $\bigoplus_k$ the *power operator*. For any integer $n \geq 2$, the operator $\biguplus_n$ is n-ary; it takes as input a sequence of elements $a_1, a_2, \ldots, a_n$, and it yields the result $\biguplus_n (a_1, a_2, \ldots, a_n) = a_1 \oplus a_2 \oplus \ldots \oplus a_n$, denoted as $\biguplus_{i=1}^{n} a_i$. (By associativity, the result is well defined.) Call $\biguplus$ the *summa-*

tion operator.

Monotone Groups. Assume now that the set $\mathbf{\Gamma}$ is totally ordered; thus, a *total order* $\preceq$ is defined on $\mathbf{\Gamma}$. For any pair of elements $a, b \in \mathbf{\Gamma}$, write $a \prec b$ if $a \preceq b$ and $a \neq b$. A *monotone subdomain* of $\mathbf{\Gamma}$ is a subset $\mathbf{M} \subseteq \mathbf{\Gamma}$ that satisfies the following three properties: *(1) Closure:* for any two elements $a, b \in \mathbf{M}$, $a \oplus b \in \mathbf{M}$, *(2) Identity Lower Bound:* for any element $a \in \mathbf{M}$, $e \prec a$, and *(3) Monotonicity under Composition:* for any pair of elements $a, b \in \mathbf{M}$, both $a \prec a \oplus b$ and $b \prec a \oplus b$. Notice that $e \notin \mathbf{M}$. Notice also that $\mathbf{M}$ is necessarily infinite. A *monotone group* is a quadruple $\langle \mathbf{\Gamma}, \mathbf{M}, \oplus, \preceq \rangle$, where $\langle \mathbf{\Gamma}, \oplus \rangle$ is an Abelian group, $\preceq$ is a total order on $\mathbf{\Gamma}$, and $\mathbf{M} \subseteq \mathbf{\Gamma}$ is a monotone subdomain of $\mathbf{\Gamma}$.

We proceed with some examples of monotone groups that will be used in our later analysis. Throughout, denote Z, $\mathbf{N}$ and Q the sets of integers, natural numbers (including zero), and rational numbers, respectively. We will use $+$ and $\cdot$ to denote the common (binary) operators of addition and multiplication, respectively, on these sets. Denote $\leq$ the *less-than-or-equal* relation (total order) on these sets. The quadruple $\langle \mathrm{Z}, \mathbf{N} \setminus \{0\}, +, \leq \rangle$ is a monotone group (*integers with addition*). From the definition of the power operator $\bigoplus_k$, for any integer k, we have that for any integer $a \in \mathrm{Z}$, $\bigoplus_k a = k \cdot a$. From the definition of the summation operator $\biguplus_{k_1}^{k_2}$, for any pair of integers k_1 and k_2, we have that for any sequence of $k_2 - k_1 + 1$ integers $a_{k_1}, a_{k_1+1}, \ldots, a_{k_2} \in \mathrm{Z}$, $\biguplus_{i=k_1}^{k_2} a_i = \sum_{i=k_1}^{k_2} a_i$. The quadruple $\langle \mathrm{Q}, \mathbf{N} \setminus \{0, 1\}, \cdot, \leq \rangle$ is also a monotone group (*integers with multiplication*). From the definition of the power operator $\bigoplus_k$, for any integer k, we have that for any rational number $a \in \mathrm{Q}$, $\bigoplus_k a = a^k$. From the definition of the summation operator $\biguplus$, for any set of n integers $k_1, k_2, \ldots, k_n$, we have that for any set of n rational numbers $a_{k_1}, a_{k_1+1}, \ldots, a_{k_2} \in \mathrm{Q}$, $\biguplus_{i=k_1}^{k_2} a_i = \prod_{i=k_1}^{k_2} a_i$.

Independence. Fix any integer $n \geq 2$, and consider any n distinct elements $a_1, a_2, \ldots, a_n \in \mathbf{\Gamma}$ with $a_1, a_2, \ldots, a_n \neq e$. Say that $a_1, a_2, \ldots, a_n$ are *n-wise independent in* $\langle \mathbf{\Gamma}, \oplus \rangle$ if for any sequence of n integers $k_1, k_2, \ldots, k_n$, where $-1 \leq k_i \leq 2$ for $1 \leq i \leq n$, that are *not all simultaneously* zero, $\biguplus_{i=1}^{n} \bigoplus_{k_i} a_i \neq e$. Say that the monotone group $\langle \mathbf{\Gamma}, \mathbf{M}, \oplus, \preceq \rangle$ is *n-wise independent* if there are n distinct elements $a_1, a_2, \ldots, a_n \in \mathbf{M}$, which are n-wise independent in $\langle \mathbf{M}, \oplus \rangle$.

By the definition of n-wise independence, n integers $a_1, a_2, \ldots, a_n \in \mathbf{N} \setminus \{0\}$, where $n \geq 2$, are n-wise independent in $\langle \mathbf{N} \setminus \{0\}, + \rangle$ if for any sequence of n integers $k_1, k_2, \ldots, k_n \in \{-1, 0, 1, 2\}$, that are not all simultaneously zero, $\sum_{i=1}^{n} k_i a_i \neq 0$. We are able to prove that for any integer $n \geq 2$, the monotone group $\langle \mathrm{Z}, \mathbf{N} \setminus \{0\}, +, \leq \rangle$ is n-wise independent. From the definition of n-wise independence, n integers $a_1, a_2, \ldots, a_n \in \mathbf{N} \setminus \{0, 1\}$ are n-wise independent in $\langle \mathbf{N} \setminus \{0, 1\}, \cdot \rangle$ if for any sequence of n integers $k_1, k_2, \ldots, k_n \in \{-1, 0, 1, 2\}$, that are not all simultaneously zero, $\prod_{i=1}^{n} a_i^{k_i} \neq 1$. Consider any n distinct prime numbers $a_1, a_2, \ldots, a_n$.

Then, $\prod_{i=1}^{n} a_i^{k_i}$ is a rational number whose numerator and denominator have no common factors; so $\prod_{i=1}^{n} a_i^{k_i} \neq 1$, and the n integers $a_1, a_2, \ldots, a_n$ are n-wise independent in $\mathbf{N} \setminus \{0,1\}$. This implies that the monotone group $\langle Q, \mathbf{N} \setminus \{0,1\}, \cdot, \leq \rangle$ is n-wise independent.

Independence of Monotone Groups. We are able to show that *every* monotone group is n-wise independent. The proof uses a reduction to the (already proven) n-wise independence of the monotone group $\langle Q, \mathbf{N} \setminus \{0,\}, +, \leq \rangle$. (Thus, this establishes some kind of *completeness* of the monotone group $\langle Q, \mathbf{N} \setminus \{0\}, +, \leq \rangle$ for the class of all n-wise independent monotone groups.)

3 System Model

Systems that Implement Monotone Groups. Our model of a distributed system is patterned after the one in [12, Section 2], adjusted to incorporate the issue of implementing a monotone group $\langle \mathbf{\Gamma}, \mathbf{M}, \oplus, \preceq \rangle$. We consider a distributed system $\mathbf{P}$ consisting of a collection of sequential threads of control, called *processes*. Processes are sequential, and each process applies a sequence of operations to a distributed data structure, called the *object*, alternately issuing an invocation and then receiving the associated response. Each *invocation* at process p_i has the form $\texttt{Invoke}_i(a)$ for some value $a \in \mathbf{M}$; each *response* at process p_i has the form $\texttt{Response}_i(b)$ for some value $b \in \mathbf{M} \cup \{e\}$. Formally, an *execution* of system $\mathbf{P}$ is a (possibly infinite) sequence α of *invocation* and *response* events. We assume that for each invocation at process p_i in execution α, there is a later response in α that matches it and no invocation at p_i that precedes the matching response in α. An *operation* at process p_i in execution α is a matching pair $op_i = [\texttt{Invoke}_i(a), \texttt{Response}_i(b)]$ of an invocation and response at p_i; for such an operation, we will write $a = \mathsf{In}(op_i)$ and $b = \mathsf{Out}(op_i)$, and we will sometimes say that op_i is of *type* a.

An execution α induces a partial order $\xrightarrow{\alpha}$ on the set of operations in α as follows: For any two operations $op_{i_1} = [\texttt{Invoke}_{i_1}(a_1), \texttt{Response}_{i_1}(b_1)]$ and $op_{i_2} = [\texttt{Invoke}_{i_2}(a_2), \texttt{Response}_{i_2}(b_2)]$ at processes p_{i_1} and p_{i_2}, respectively, say that op_{i_1} *precedes* op_{i_2} *in execution* α, denoted $op_{i_1} \xrightarrow{\alpha} op_{i_2}$, if the response $\texttt{Response}_{i_1}(b_1)$ precedes the invocation $\texttt{Invoke}_{i_2}(a_2)$. In particular, execution α induces, for each process p_i a total order $\xrightarrow{\alpha}_i$ on the set of operations at p_i in α as follows: For any two operations $op_i^{(1)}$ and $op_i^{(2)}$, $op_i^{(1)} \xrightarrow{\alpha}_i op_i^{(2)}$ if and only if $op_i^{(1)} \xrightarrow{\alpha} op_i^{(2)}$. If, in execution α, operation op_{i_1} does not precede operation op_{i_2}, then we write $op_{i_1} \not\xrightarrow{\alpha} op_{i_2}$. If simultaneously $op_{i_1} \not\xrightarrow{\alpha} op_{i_2}$ and $op_{i_2} \not\xrightarrow{\alpha} op_{i_1}$, then we say that op_{i_1} and op_{i_2} are *parallel* in execution α, denoted as $op_{i_1} \|_\alpha op_{i_2}$.

For any execution α of system $\mathbf{P}$, a *serialization* $S(\alpha)$ of execution α is a se-

quence whose elements are the operations of α, and each operation of α appears exactly once in $S(\alpha)$. Thus, a serialization $S(\alpha)$ is a total order $\xrightarrow{S(\alpha)}$ on the set of operations in α. Notice that there may be, in general, many possible serializations of the execution α. Say that a serialization $S(\alpha)$ is *valid for the monotone group* $\langle \mathbf{\Gamma}, \mathbf{M}, \oplus, \preceq \rangle$ if the following two conditions hold: *(1) Valid Start:* if $op_i = [\texttt{Invoke}_i(a), \texttt{Response}_i(b)]$ is the first operation in $S(\alpha)$, then $b = e$, and *(2) Valid Composition:* for any pair of operations $op_{i_1}^{(1)} = [\texttt{Invoke}_{i_1}(a_1), \texttt{Response}_{i_1}(b_1)]$ and $op_{i_2}^{(2)} = [\texttt{Invoke}_{i_2}(a_2), \texttt{Response}_{i_2}(b_2)]$ that are consecutive in $S(\alpha)$, $b_2 = b_1 \oplus a_1$. Say that $\mathbf{P}$ *implements the monotone group* $\langle \mathbf{\Gamma}, \mathbf{M}, \oplus, \preceq \rangle$ if every execution of α has a valid serialization. We prove the *Unique Serialization Lemma*, asserting that for any execution α of $\mathbf{P}$ implementing a monotone group, there is a *unique* valid serialization $S(\alpha)$. Sometimes, we will write $\mathsf{In}_\alpha(op)$ and $\mathsf{Out}_\alpha(op)$ in order to emphasize reference to execution α.

Linearizable Executions. We consider a system $\mathbf{P}$ that implements a monotone group $\langle \mathbf{\Gamma}, \mathbf{M}, \oplus, \preceq \rangle$. Say that execution α is *linearizable* [12] if the (unique) valid serialization $S(\alpha)$ extends $\xrightarrow{\alpha}$; that is, for any pair of operations $op^{(1)}$ and $op^{(2)}$ such that $op^{(1)} \xrightarrow{\alpha} op^{(2)}$, $op^{(1)} \xrightarrow{S(\alpha)} op^{(2)}$. Since $\mathbf{P}$ implements the monotone group $\langle \mathbf{\Gamma}, \mathbf{M}, \oplus, \preceq \rangle$, for any two operations $op^{(1)}$ and $op^{(2)}$ such that $op^{(1)} \xrightarrow{S(\alpha)} op^{(2)}$, $\mathsf{Out}(op^{(1)}) \prec \mathsf{Out}(op^{(2)})$. Thus, it follows that for any pair of operations $op^{(1)}$ and $op^{(2)}$ such that $op^{(1)} \xrightarrow{\alpha} op^{(2)}$, $\mathsf{Out}(op^{(1)}) \prec \mathsf{Out}(op^{(2)})$. Say that operation $op^{(1)}$ in execution α is *non-linearizable in execution* α if there is another operation $op^{(2)}$ in execution α such that $op^{(2)} \xrightarrow{\alpha} op^{(1)}$ while $\mathsf{Out}(op^{(1)}) \prec \mathsf{Out}(op^{(2)})$. Say that operation op in execution α is *linearizable in execution* α if it is not non-linearizable in execution α. Clearly, execution α is linearizable if every operation in execution α is linearizable in it.

4 Switching Networks

Basic Definitions. A *switching network* [7] is a directed acyclic graph in which the nodes are called *switches* and the edges are called *wires*. An (f_{in}, f_{out})*-switch* is a routing element with f_{in} *input wires*, f_{out} *output wires*, and an internal *state*. A (w_{in}, w_{out})*-switching network* has w_{in} *input wires* and w_{out} *output wires*, and it is formed by connecting together switches; thus, we connect output wires of switches to input wires of other switches. Some switches have input wires (resp., output wires) not connected to other switches in the network, and these wires are called the *input wires* (resp., *output wires*) of the network. The *size* of a switching network is the number of its switches. A *path* in a switching network is a sequence of switches, each connected to the next. The *depth* $d(b)$ of a switch b in a switching network is defined to be 0 if one of its input wires is an input wire of

the network, and $\max_j d(b_j) + 1$, where the maximum is taken over all switches b_j that are connected *to* switch b. The *depth* d of the network is defined as the maximum depth of any switch. The switching network can naturally be divided into d *layers,* so that layer ℓ contains all switches of depth ℓ, where $0 \leq \ell \leq d$.

Tokens. Processes access the switching network by issuing *tokens*. In contrast to counting networks [2], each token has a *state* (a set of variables) which can change as the token traverses the network. In particular, a token enters the switching network from one of the network's input wires; then, the token is forwarded to the switch to which the wire belongs, the switch then routes the token to one of its output wires from which the token enters the next switch in the network, and so on. The token continues traversing the network in the same fashion until it reaches an output wire of the network. Then, the token exits the network and returns to the process that issued it. When a token traverses a switch, the states of the token and the switch change atomically before the token is routed to an output wire of the switch. Note that the token and the switch have different transition functions for their states. A switching network may be accessed by many tokens simultaneously which traverse the network asynchronously; however, each process has at most one token traversing the network each time. The *latency* of the switching network is the maximum number of switches traversed by any token (thus, it does not exceed the depth of the network). The *concurrency* of a switching network is the maximum number of processes (hence, tokens) allowed to access the network simultaneously.

Configurations. A *network configuration* of a switching network is the concatenation of the current states of the network's switches. A *total configuration* of a switching network is the concatenation of the current states of the networks' switches and the states of all tokens that are currently traversing the network. Say that a switching network is in a *quiescent* total configuration if there are no tokens traversing the network (that is, all tokens that have entered the network have exited it). Denote x_i the total number of tokens that have ever entered from input wire i of the network, where $1 \leq i \leq w_{in}$, and denote y_j the total number of tokens that have left from output wire j of the network, where $1 \leq j \leq w_{out}$. The network must satisfy the following two properties: *(1) Safety property:* in any total configuration, it must be $\sum_{i=1}^{w_{in}} x_i \geq \sum_{j=1}^{w_{out}} y_j$; thus, no new tokens are created in the network, and *(2) Liveness Property:* given any finite number of input tokens that traverse the network, the network will eventually reach a quiescent total configuration. In any quiescent configuration it must be that $\sum_{i=1}^{w_{in}} x_i = \sum_{j=1}^{w_{out}} y_j$. The safety and liveness properties must also be satisfied by every individual switch in the network.

Executions. We model executions of switching networks in the style of Herlihy *et al.* [10]. For any switch b and token t, we denote by $e = \langle t, b \rangle$ the *transition* in which the token passes (in one atomic step) from an input wire to an output wire of switch b. An *execution* of a switching network is a finite or infinite sequence $s_0, e_1, s_1, e_2, \ldots$ of alternating total configurations and switch transitions such that for each triple $\langle s_i, e_{i+1}, s_{i+1} \rangle$, the switch transition e_{i+1} carries the total configuration s_i to total configuration s_{i+1}. A finite execution is *complete* if it results to a quiescent total configuration. An execution α is *sequential* if for any two transitions $e_i = \langle t_i, b_i \rangle$ and $e_j = \langle t_j, b_j \rangle$, where t_i and t_j correspond to the same token, all transitions (if any) between them also involve that token. In other words, tokens traverse the network one completely after the other in a sequential execution. In an execution of a switching network, we say that concurrency is *bounded* if the number of concurrent processes accessing the network in the execution is bounded. In an (infinite) execution, we say that concurrency is *unbounded* if the number of concurrent processes accessing the network in the execution is unbounded (in which case it is either finite or infinite).

Implementations. A switching network $\mathcal{N}$ can be used to implement a monotone group $\langle \Gamma, \mathbf{M}, \oplus, \preceq \rangle$. Each token t issued by process p_i corresponds to an operation $op_i = [\texttt{Invoke}_i(a), \texttt{Response}_i(v)]$ invoked by process p_i, where $a \in \mathbf{M}$ and $v \in \mathbf{M} \cup \{e\}$. We say that a is the *input value* of the token t, and v is the *output value* of the token t. The input value of the token is part of the token's initial state. In any execution α, the invocation of operation op corresponds to the first transition $e_i = \langle t_i, b_i \rangle$ where $t_i = t$ and b_i is an input switch of the network (this transition occurs when the token enters the network); the response of op corresponds to the latest transition $e_j = \langle t_j, b_j \rangle$ in execution α such that $t_j = t$ (this transition occurs when the token exits the network). When token t exits the network, it carries encapsulated in its state the output value v that operation op_i responds with. Use execution α to define its subsequence α' that contains only transitions that correspond to invocations and responses of the operations corresponding to tokens. The sequence α' induces an execution of a distributed system in the natural way. Denote $\mathbf{P}$ the distributed system that is determined by all such induced executions (one for each execution of the switching network $\mathcal{N}$). Say now that *switching network $\mathcal{N}$ implements the monotone group* $\langle \Gamma, \mathbf{M}, \oplus, \preceq \rangle$ if the system $\mathbf{P}$ implements the monotone group $\langle \Gamma, \mathbf{M}, \oplus, \preceq \rangle$.

Finite and Infinite Switches. We examine two kinds of switching networks, corresponding to switches with finite or infinite state.

- *Switching networks with finite switches:* Each switch of the network has a *finite* number of states. For this kind of network, we include an additional component on the output wires of the switching network: the *output registers*. There is an output register associated with each output wire of the

switching network. Unlike switches, each output register has an infinite number of states. The output value for a token's operation is computed on the output register residing on the network's output wire from which the token exits. At the exit, the following happen atomically: the token computes its output value according to the register's current state and the state of the register changes according to its previous state and the state of the token (which includes its input value). Notice that the input value of a token does not affect its output value, but only the output value of the next token that will access the same output register.

We remark that this kind of switching networks corresponds more to traditional counting networks [2], where a token fetching the counter's value and incrementing the counter by one obtains the value from the register attached to the output wire it will arrive at. We also remark that output registers are *necessary* for this kind of switching networks, since they provide an infinite number of different output values to tokens, while finite switches, used only for routing, are unable to do so.

- *Switching networks with infinite switches:* Each switch has an infinite number of states. For this kind of networks, there are no attached output registers and the output value of a token is determined according to the state of the token when it exits the network.

Contention Measures. In a switching network, contention represents the extent to which concurrent processes access the same switch or output register simultaneously. We use the following complexity-theoretic measures to model contention in switching networks, the last of which was originally introduced by Dwork *et al.* [6] for the case of counting networks.

- The *register bottleneck* of a switching network $\mathcal{N}$ is the *minimum* number of output registers, the minimum being taken over all infinite executions, accessed by tokens in an infinite suffix of an infinite execution of $\mathcal{N}$. (This definition applies only to switching networks with finite switches.) Intuitively, a switching network is low-contention if its register bottleneck is large; a register bottleneck of 1 is the *worst,* since then many tokens (as many as processes) may eventually accumulate in front of the same output register, which becomes a "hot-spot".
- Similarly, we define the *layer bottleneck* of a switching network $\mathcal{N}$ to be the *minimum* number of switches in the same layer, the minimum being taken over all layers and infinite executions, accessed by tokens in an infinite suffix of an infinite execution of $\mathcal{N}$. (This definition will be useful for switching networks with infinite switches.) Intuitively, a switching network is low-contention if its layer bottleneck is large; a layer bottleneck of 1 is the *worst,* since then many tokens (as many as processes) may eventually accumulate in front of the same switch, which becomes a "hot-spot".

- The *capacity* c of a switching network $\mathcal{N}$ is the maximum number of processes that simultaneously access a particular switch in any execution of $\mathcal{N}$.

5 The Monotone Linearizability Lemma

In this section, we state and prove the Monotone Linearizability Lemma, which establishes ordering constraints of linearizability on a system **P** that implements a monotone group $\langle \mathbf{\Gamma}, \mathbf{M}, \oplus, \preceq \rangle$. Since the monotone group $\langle \mathbf{\Gamma}, \mathbf{M}, \oplus, \preceq \rangle$ is n-wise independent, there exist n distinct elements $a_1, a_2, \ldots, a_n \in \mathbf{M}$, with $a_1, a_2, \ldots, a_n \neq e$, which are n-wise independent in $\langle \mathbf{M}, \oplus \rangle$. The proof of the Monotone Linearizability Lemma amounts to establishing a contradiction to n-wise independence for a hypothetical *non-linearizable* execution, in which the arguments of the RMW operations issued by the processes are $a_1, a_2, \ldots, a_n$. We show:

Proposition 1 (Monotone Linearizability Lemma) *Consider any execution* α *of system* **P** *in which each process* p_i, $1 \leq i \leq n$, *issues only operations of type* a_i. *Then,* α *is linearizable.*

6 Impossibility Results and Lower Bounds

Lower Bounds on Size. We first consider switching networks with finite switches. We show:

Theorem 1 (Switching Networks with Finite Switches) *There is no non-trivial switching network with finite switches that has finite size, incurs register bottleneck at least* 2 *and implements a monotone group* $\langle \mathbf{\Gamma}, \mathbf{M}, \oplus, \preceq \rangle$, *when the concurrency is bounded.*

We remark that the concurrency assumed in the proof of Theorem 1 is no more than the number of tokens involved in the proof, which is a *bounded* quantity depending only on parameters of the network $\mathcal{N}$. Thus, the impossibility result in Theorem 1 holds even for networks with bounded concurrency. Finally, we argue that the assumption of a *non-trivial* switching network is essential for Theorem 1 to hold: since each token can atomically invoke a computation on an output register, we can implement a monotone RMW operation by a *trivial* switching network consisting of a single switch that outputs tokens along one output wire, which has an associated register that maintains the state of the RMW variable to be implemented. The switch sequences the operations (that correspond to the tokens) so that they can be atomically invoked (by the tokens) on the register.

We now turn to switching networks with infinite switches. Clearly, the proof of Theorem 1 is not applicable to switching networks with infinite switches, since

the number of their possible network configurations is no longer finite. Thus, we need to develop new techniques to handle them. We show:*

Theorem 2 (Switching Networks with Infinite Switches) *There is no non-trivial switching network with infinite switches that has finite size, incurs layer bottleneck at least 2 and implements a monotone group* $\langle \mathbf{\Gamma}, \mathbf{M}, \oplus, \preceq \rangle$, *assuming concurrency is unbounded.*

We remark that the proof of Theorem 2 requires unbounded (finite or infinite) concurrency. So, Theorem 2 does *not* imply Theorem 1 which assumes bounded concurrency, and the two results are incomparable and represent a trade-off. Finally, we remark that the assumption of a *non-trivial* switching network is essential for Theorem 2 to hold: A switching network consisting of a single infinite-state switch with n input wires and n output wires (where n is the number of concurrent processes) can implement any RMW variable as follows. The state of the variable is encoded by the state of the switch. To invoke an operation on the variable, a process issues a token with a state encoding the argument of the operation. Such a token, when atomically processed by the switch, will cause the natural changes to its state and to the state of the switch, so that the new state of the switch is the new state of the variable, and the new state of the token is the response of the variable to the operation invoked by the token.

Lower Bound on Time. We start with a definition that we will use in our proof. For any quiescent total configuration s of a switching network $\mathcal{N}$, we say that token t_i has *preferred path* π if t_i follows the path π and runs in isolation into the network, which is initially in the total configuration s, until token t_i exits the network and responds with an output value v which is its *preferred value*. We show:

Theorem 3 *For any switching network* $\mathcal{N}$ *that implements a monotone group* $\langle \mathbf{\Gamma}, \mathbf{M}, \oplus, \preceq \rangle$, *there is a sequential execution with n tokens such that each token traverses at least* $\lceil \frac{n-1}{c-1} \rceil$ *switches.*

7 Conclusion and Open Problems

We have studied the possibility, and the corresponding costs, of implementing a monotone RMW operation in a concurrent and low-contention manner. Our end

*The proof of Theorem 2 will consider (without loss of generality) so called *normalized* switching networks, in which any switch b at layer ℓ has its input wires connected to switches of layer $\ell - 1$ (assuming $\ell \geq 2$) and its output wires connected to switches of layer $\ell + 1$ (assuming ℓ is less than the depth of the network). Thus, in a normalized switching network, there are no wires connecting switches in non-consecutive layers. Note that any switching network can be easily cast as a normalized one, if we intercept wires that connect non-consecutive layers with *dummy* switches with input and output width 1, which simply forward tokens (without routing them).

results are lower bounds on size and latency for any non-trivial, low-contention switching network that implements a monotone RMW operation; these are shown by using the Monotone Linearizability Lemma, which may be of independent interest. It would be interesting to ask whether *timing conditions* may suffice to overcome the limitations we have shown; recall that timing conditions have been exploited in the work of Lynch *et al.* [16] for devising *finite-size* linearizable counting networks, while Herlihy *et al.* [10] establish that no finite-size (non-trivial) *asynchronous* linearizable counting network exists. For future work, we are also interested in establishing further limitations on various kinds of distributed systems (other than switching networks) that implement a monotone RMW operation. A natural candidate to consider is the message-passing system adopted in the work by Wattenhofer and Widmayer [19]; that work showed a lower bound on the message complexity of implementing the Fetch&Increment operation in that system; we feel that similar limitations hold for implementations of *any* monotone RMW operation.

References

[1] W. Aiello, C. Busch, M. Herlihy, M. Mavronicolas, N. Shavit and D. Touitou, "Supporting Increment and Decrement Operations in Balancing Networks," *Chicago Journal of Theoretical Computer Science*, 2000-4, December 14, 2000 (electronic). Preliminary version in *Proceedings of the 16th International Symposium on Theoretical Aspects of Computer Science (STACS 1999)*, Vol. 1563, Lecture Notes in Computer Science, Springer-Verlag, pp. 377–386, 1999.

[2] J. Aspnes, M. Herlihy and N. Shavit, "Counting Networks," *Journal of the ACM*, Vol. 41, pp. 1020–1048, 1994.

[3] H. Attiya and J. L. Welch, *Distributed Computing: Fundamentals, Simulations and Advanced Topics*, McGraw-Hill, 1998.

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

[5] C. Busch and M. Mavronicolas, "A Combinatorial Treatment of Balancing Networks," *Journal of the ACM*, Vol. 43, No. 5, pp. 794–839, September 1996.

[6] C. Dwork, M. Herlihy and O. Waarts, "Contention in Shared Memory Algorithms," *Journal of the ACM*, Vol. 44, No. 6, pp. 779–805, November 1997.

[7] P. Fatourou and M. Herlihy, "Adding Networks," *Proceedings of the 15th International Symposium on DIStributed Computing (DISC 2001)*, J. L. Welch ed., pp. 330–342, Vol. 2180, Lecture Notes in Computer Science, Springer-Verlag, Lisbon, Portugal, October 2001.

[8] J. Goodman, M. Vernon and P. Woest, "Efficient Synchronization Primitives for Large-Scale, Cache-Coherent Multiprocessors," *Proceedings of the 3rd International*

Conference on Architectural Support for Programming Languages and Operating Systems, pp. 64–75, April 1989.

[9] M. Hall, Jr., *Theory of Groups,* Second Edition, Chelsea Publishing Company, 1979.

[10] M. Herlihy, N. Shavit and O. Waarts, "Linearizable Counting Networks," *Distributed Computing,* Vol. 9, pp. 193–203, 1996.

[11] M. Herlihy, S. Tirthapura and R. Wattenhofer, "Ordered Multicast and Distributed Swap," *Operating Systems Review,* Vol. 35, No. 1, pp. 85–96, January 2001.

[12] 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, 1990.

[13] C. P. Kruskal, L. Rudolph and M. Snir, "Efficient Synchronization on Multiprocessors with Shared Memory," *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing,* pp. 218–228, August 1986.

[14] 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.

[15] N. Lynch, *Distributed Algorithms,* Morgan Kaufmann, 1996.

[16] 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. 67–91, June 1999.

[17] M. Mavronicolas, M. Merritt and G. Taubenfeld, "Sequentially Consistent versus Linearizable Counting Networks," *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing,* pp. 133–142, May 1999.

[18] N. Shavit and D. Touitou, "Elimination Trees and the Construction of Pools and Stacks," *Theory of Computing Systems,* Vol. 30, No. 6, pp. 545–570, November/December 1997.

[19] R. Wattenhofer and P. Widmayer, "An Inherent Bottleneck in Distributed Counting," *Journal of Parallel and Distributed Computing,* Vol. 49, pp. 135–145, 1998.

Costas Busch is with Rensselaer Polytechnic Institute, Troy, NY 12180 USA. E-mail: buschc@cs.rpi.edu

Marios Mavronicolas is with the University of Cyprus, 1678 Nicosia, Cyprus. E-mail: mavronic@ucy.ac.cy

Paul Spirakis is with the University of Patras, Rion, 265 00 Patras, Greece and the Computer Technology Institute, P. O. Box 1122, 261 10 Patras, Greece. E-mail: spirakis@cti.gr

results are lower bounds on size and latency for any non-trivial, low-contention switching network that implements a monotone RMW operation; these are shown by using the Monotone Linearizability Lemma, which may be of independent interest. It would be interesting to ask whether *timing conditions* may suffice to overcome the limitations we have shown; recall that timing conditions have been exploited in the work of Lynch *et al.* [16] for devising *finite-size* linearizable counting networks, while Herlihy *et al.* [10] establish that no finite-size (non-trivial) *asynchronous* linearizable counting network exists. For future work, we are also interested in establishing further limitations on various kinds of distributed systems (other than switching networks) that implement a monotone RMW operation. A natural candidate to consider is the message-passing system adopted in the work by Wattenhofer and Widmayer [19]; that work showed a lower bound on the message complexity of implementing the Fetch&Increment operation in that system; we feel that similar limitations hold for implementations of *any* monotone RMW operation.

References

[1] W. Aiello, C. Busch, M. Herlihy, M. Mavronicolas, N. Shavit and D. Touitou, "Supporting Increment and Decrement Operations in Balancing Networks," *Chicago Journal of Theoretical Computer Science*, 2000-4, December 14, 2000 (electronic). Preliminary version in *Proceedings of the 16th International Symposium on Theoretical Aspects of Computer Science (STACS 1999)*, Vol. 1563, Lecture Notes in Computer Science, Springer-Verlag, pp. 377–386, 1999.

[2] J. Aspnes, M. Herlihy and N. Shavit, "Counting Networks," *Journal of the ACM*, Vol. 41, pp. 1020–1048, 1994.

[3] H. Attiya and J. L. Welch, *Distributed Computing: Fundamentals, Simulations and Advanced Topics*, McGraw-Hill, 1998.

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

[5] C. Busch and M. Mavronicolas, "A Combinatorial Treatment of Balancing Networks," *Journal of the ACM*, Vol. 43, No. 5, pp. 794–839, September 1996.

[6] C. Dwork, M. Herlihy and O. Waarts, "Contention in Shared Memory Algorithms," *Journal of the ACM*, Vol. 44, No. 6, pp. 779–805, November 1997.

[7] P. Fatourou and M. Herlihy, "Adding Networks," *Proceedings of the 15th International Symposium on DIStributed Computing (DISC 2001)*, J. L. Welch ed., pp. 330–342, Vol. 2180, Lecture Notes in Computer Science, Springer-Verlag, Lisbon, Portugal, October 2001.

[8] J. Goodman, M. Vernon and P. Woest, "Efficient Synchronization Primitives for Large-Scale, Cache-Coherent Multiprocessors," *Proceedings of the 3rd International*