Tight Bounds for Shared Memory Systems Accessed by Byzantine Processes*

Noga Alon† Michael Merritt‡ Omer Reingold§ Gadi Taubenfeld¶
Rebecca N. Wright‖

June 15, 2005

Abstract

We provide efficient constructions and tight bounds for shared memory systems accessed by n processes, up to t of which may exhibit Byzantine failures, in a model previously explored by Malkhi et al. [MMRT03]. We show that sticky bits are universal in the Byzantine failure model for $n \geq 3t+1$, an improvement over the previous result requiring $n \geq (2t+1)(t+1)$. Our result follows from a new strong consensus construction that uses sticky bits and tolerates t Byzantine failures among n processes for any $n \geq 3t+1$, the best possible bound on n for strong consensus. We also present tight bounds on the efficiency of implementations of strong consensus objects from sticky bits and similar primitive objects.

1 Introduction

Although Byzantine fault tolerance in message-passing systems has been extensively investigated, it was only recently that Malkhi et al. initiated the study of Byzantine fault tolerance in asynchronous shared memory systems [MMRT03]. Their work establishes a formal model and shows how the use of access control lists (ACLs) can constrain Byzantine behavior and permit reliable distributed computation. They investigate *universal* objects, which can be used to implement any shared object. Specifically, they show that sticky bits, a simple shared memory primitive long known to be universal in the crash failure model [Plo89], are also universal in the Byzantine failure model, provided that $n \geq (2t+1)(t+1)$, where n is the number of processes and t bounds the number of processes that may fail (exhibiting unconstrained, or Byzantine, behavior). One of the main results of this paper is to strengthen their result, showing that sticky bits are universal in the Byzantine failure model for any $n \geq 3t+1$.

The universality results of [MMRT03] first use constructions from sticky bits to build strong consensus objects, then use strong consensus objects in an explicit universal construction of an arbitrary shared object. (Definitions of sticky bits, weak and strong consensus, and other objects mentioned in this introduction are

*A preliminary version of the results presented in this paper appeared in [MRTW02].

†Schools of Mathematics and Computer Science, Tel Aviv University, Tel Aviv, Israel, email: nogaa@tau.ac.il. Research supported in part by a grant from the Israel Science Foundation, and by the Hermann Minkowski Minerva Center for Geometry at Tel Aviv University.

‡AT&T Labs, 180 Park Ave., Florham Park, NJ 07932, USA. mischu@research.att.com

§Incumbent of the Walter and Elise Haas Career Development Chair. Dept. of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel. omer.reingold@weizmann.ac.il. This work was partially completed while at AT&T Labs and while visiting the Institute for Advanced Study, Princeton, NJ. Research supported in part by US-Israel Binational Science Foundation Grant 2002246.

¶The School of Computer Science, the Interdisciplinary Center, P.O.Box 167, Herzliya 46150, Israel. tgadi@idc.ac.il. This work was partially completed while visiting AT&T Labs.

‖Computer Science Dept., Stevens Institute of Technology, Hoboken, NJ 07030, USA. rwright@cs.stevens.edu. This work was partially completed while at AT&T Labs. Research supported in part by the National Science Foundation under Grant No. CCR-0331584.

provided in Section 2.) The bound $n \geq (2t+1)(t+1)$ follows from the construction in the first step, building strong consensus from sticky bits. In this paper, we present a novel construction of strong consensus from sticky bits for any $n \geq 3t+1$. The consequence for universality is immediate: "Constructions of strong consensus from sticky bits for larger values of t would imply a more resilient universality result." [MMRT03]. Malkhi et al. demonstrate that strong consensus objects can only exist if $n \geq 3t+1$, so our result is the best possible unless a different universal construction is used. Beyond strengthening the universality result for sticky bits, we present tight bounds on the efficiency of implementations of strong consensus objects from sticky bits and similar "potentially powerful" shared objects. (Potentially powerful objects include sticky bits that can be set by more than one process, but exclude registers and single-writer sticky bits. Formally, potentially powerful operations are those other than wait-free reads and writes.)

In Section 2, we review the model and definitions. Section 3 presents a general protocol schema that can be used to implement strong consensus from sticky bits. Instantiated separately for $n = 3t+1$ and $n = (t+1)^2$, the schema results in two strong consensus algorithms, which use $\binom{2t+1}{t}$ and $t+1$ sticky bits, correspondingly. The contrast in efficiency of these constructions is striking: exponentially many potentially powerful objects for $n = 3t+1$ and just $t+1$ for $n = (t+1)^2$. We then modify the latter protocol to show that t sticky bits are sufficient, provided $n \geq t^2 + 5t + 1$. Surprisingly, this algorithm works despite the fact that all of the t potentially powerful shared objects could be written by t Byzantine processes.

In Section 4, we demonstrate bounds and tradeoffs among n, t, and the number and type of objects used. We show that the protocols of Section 3 are essentially optimal in the number of potentially powerful shared objects used. In Section 4.1, we present a general lower bound that constrains the types of potentially powerful shared objects and associated access control lists required to implement even weak consensus. Sections 4.2 and 4.3 investigate the tradeoff on the number of potentially powerful shared objects that are necessary and sufficient for constructing strong consensus objects as n increases relative to t. The sufficiency results are constructive, providing explicit instantiations of the protocol schema of Section 3.

1.1 Summary of results

To summarize, the main contributions of this paper are the following:

- We present a strong consensus protocol that can tolerate t Byzantine failures (and therefore a universality result) from sticky bits, for $n \geq 3t+1$.
- We present a strong consensus protocol that can tolerate t Byzantine failures using only t potentially powerful shared objects. (Surprising since t Byzantine processes can access all the potentially powerful shared objects.)
- We show that at least t shared objects (such as sticky bits) must be used in strong consensus protocols that can tolerate t Byzantine failures.
- We show that any weak consensus protocol that tolerates t crash failures must use at least one object on which at least $t+1$ processes can invoke potentially powerful operations.
- We demonstrate a tight tradeoff characterizing the number—as a function of n and t—of objects taken from a certain class of potentially powerful shared object that is needed to implement strong consensus. Specifically, we show that the number of potentially powerful shared objects needed to implement strong consensus is essentially $t \cdot 2^{\Theta(t^2/n)}$.
- As a consequence of the tradeoff, we obtain a polynomial-time protocol implementing strong consensus from sticky bits for $n = O(t^2/\log t)$, an improvement over the previous $n = \Omega(t^2)$ for polynomial-time strong consensus implementations from sticky bits.

1.2 Related work

The paper was inspired by the work done in [MMRT03], which includes the first study of the power of objects shared by Byzantine process. We have already discussed above many of the results that have appeared in [MMRT03].

An operation is *wait-free* if it is guaranteed to return within a finite number of steps. The power of various shared objects has been studied extensively in shared memory environments where processes may fail benignly and every operation is wait-free. Objects that can be used together with atomic registers to build wait-free implementations of any other object are called *universal objects*. Previous work on shared objects provided methods, called *universal constructions*, to transform sequential specifications of arbitrary shared objects into wait-free concurrent implementations that use universal objects [Her91, Plo89, JT92]. In particular, Herlihy proved that consensus objects are universal [Her91]. Implementing consensus with sticky bits, Plotkin then showed that sticky bits are universal [Plo89]. Herlihy also classified shared objects by their consensus number: that is, the maximum number of processes that can reach consensus using multiple instances of the object and read/write registers [Her91].

Suppose that at some point in a computation a shared register is set to some unexpected value. There are two complementary ways to explain how this may happen. One is to assume that the register's value was set by a Byzantine process, as may happen in our model. The other is to assume that the processes may be correct, but that the register itself is faulty. The subject of memory failures (as opposed to process failures) has been investigated in several papers [AGMT95, JCT98]. These papers assume any number of process crash failures, but bound the number of faulty objects, whereas we bound the number of (Byzantine) faulty processes, but each might sabotage all the objects to which it has access.

Attie investigates the power of shared objects accessed by Byzantine processes for achieving wait-free Byzantine consensus. He proves that strong consensus is impossible to achieve using objects that can be reset back to their initial setting [Att00].

The new consensus algorithms presented in this paper are based on an idea of Berman and Garay [BG89, Mis89] for performing consensus in the message passing model. The proof of one of our main impossibility results (i.e., the ACL Theorem) is inspired by proofs and techniques presented in [FLP85, LA87]; the result itself is a generalization of a result by Dwork, Herlihy, and Waarts [DHW97].

Many experimental and commercial processors provide direct support for shared memory abstractions, and increasing attention is being paid to implementing shared memory primitives and concurrent data structures either in hardware or in software [Boe04, DHLM04, Lea04, Mic04, MS04]. Our result relates to work on message-passing systems that emulate shared memory abstractions tolerant of Byzantine failures [PG89, SE+92, Rei96, KMM98, CL99, MR00]; these systems guarantee the correctness of the emulated shared objects themselves, the question is what power do these objects provide to the correct processes that use them, in the face of corrupt processes accessing them.

2 Model and definitions

The model of computation we consider was introduced by Malkhi et al., and parts of this section are adapted from [MMRT03]. This model consists of an asynchronous collection of n processes, denoted $p_1, \ldots, p_n$, that communicate via shared objects. Wait-free shared memory fault models assume no bound on the number of potentially faulty processes—each operation by a process p on a shared object must terminate, regardless of the concurrent actions of other processes. Following [MMRT03], this model differs in two ways: we make the more pessimistic assumption that process failures are Byzantine, and we make the more optimistic assumption that the number of failures is bounded by t, where t is less than the total number n of processes. In any run any process may be either *correct* or *faulty*. Correct processes are constrained to obey their specifications. A faulty processes can either crash or behave in a Byzantine way. A process that follows its protocol up to a certain point and then stops sending messages or stops accessing shared objects is called a *crashed process* or a *crash failure*. A process that deviates from its protocol either by crashing or by performing incorrect operations is called a *Byzantine process* or a *Byzantine failure*. We generally use t to

denote the maximum number of faulty processes. Whenever we discuss faulty processes we identify the type of failure assumed.

2.1 Shared objects with access control lists

Each shared object presents a set of operations. For example, $x.\mathsf{op}$ denotes operation op on object x. For each such operation $x.\mathsf{op}$ on x, there is an associated access control list, denoted $\mathrm{ACL}(x.\mathsf{op})$, which is the set of processes allowed to *invoke* that operation. Each operation execution begins with an invocation by a process in the operation's ACL, and remains pending until a response is received by the invoking process. The ACLs for two different operations on the same object can differ, as can the ACLs for the same operation on two different objects. The ACLs for an object do not change. For any operation $x.\mathsf{op}$, we say that x is k-op if $|\mathrm{ACL}(x.\mathsf{op})| = k$. A process not in the ACL for $x.\mathsf{op}$ cannot invoke $x.\mathsf{op}$, regardless of whether the process is correct or Byzantine (faulty). That is, a (correct or faulty) process cannot access an object in any way except via the operations for which it appears in the associated ACLs. Byzantine faulty processes can, for example, write different values than their specifications suggest, or refuse to invoke an operation they are supposed to invoke, but they remain constrained against invoking operations for which they are not in the ACL.

Many abstract objects support read operations: operations that return information about the state of the object, without constraining its future behavior (see [Her91]). In this paper, we assume the primitive objects (registers and sticky bits) support wait-free read operations by all processes, and focus on the ACLs for non-read operations. Atomic registers (and some other abstract objects) are *historyless* [FHS98] in that they support only operations that do not change the register value, and operations such as wait-free $\mathsf{write}()$ operations, that constrain future object behavior independently of the state in which they are invoked. These operations have long been known to be weak synchronization primitives [LA87, Her91].

We define an operation to be *potentially powerful* if it is neither a wait-free read nor a wait-free $\mathsf{write}()$ operation, and for an object (or object type) x, we define $\mathrm{ACL}_{\mathsf{pow}}(x)$ to be the union of $\mathrm{ACL}(x.\mathsf{op})$ for all potentially powerful operations $x.\mathsf{op}$ of x. Moreover, we call an object (or object type) x *potentially powerful* if $\mathrm{ACL}_{\mathsf{pow}}(x) \geq 2$. That is, potentially powerful objects are those that support potentially powerful operations by at least two different processes. Thus, neither registers nor sticky bits (defined below) writable by only one process are potentially powerful, while sticky bits writable by more than one process are potentially powerful. (Potentially powerful operations and objects such as snapshot or collect are implementable from registers yet are too weak to implement consensus. Hence we use the more accurate terminology of "potentially powerful operations and objects", instead of the shorter but misleading "powerful operations and objects" used in [MRTW02].) Our lower bounds indicate the number of potentially powerful objects required to implement weak and strong consensus. Whether such objects are sufficient will depend on their specific operation semantics.

2.2 Object definitions

Next, we define some of the types of object used in this paper.

Atomic registers: An atomic register x is an object with two operations: $x.\mathsf{read}$ and $x.\mathsf{write}(v)$ where $v \neq \perp$. An $x.\mathsf{read}$ that occurs before the first $x.\mathsf{write}()$ returns $\perp$. An $x.\mathsf{read}$ that occurs after an $x.\mathsf{write}()$ returns the value written in the last preceding $x.\mathsf{write}()$ operation.

Sticky bits: A sticky bit x is an object with two operations: $x.\mathsf{read}$ and $x.\mathsf{set}(v)$ where $v \in \{0, 1\}$. An $x.\mathsf{read}$ that occurs before the first $x.\mathsf{set}()$ returns $\perp$. An $x.\mathsf{read}$ that occurs after an $x.\mathsf{set}()$ returns the value written in the first $x.\mathsf{set}()$ operation. We will be concerned with wait-free sticky bits. (To highlight the specific semantics of the $x.\mathsf{set}()$ operation and to distinguish it from the $\mathsf{write}()$ of atomic registers, we depart from previous work [Plo89, MMRT03], which uses $\mathsf{write}()$ to denote both operations.)

Weak consensus objects: [MMRT03] A weak (binary) consensus object x is an object with one operation: $x.\mathsf{propose}(v)$, where $v \in \{0, 1\}$, satisfying: (1) In any run, the $x.\mathsf{propose}()$ operation returns the same

value, called the *consensus value*, to every correct process that invokes it. (2) In any finite run in which all participating processes are correct (no Byzantine failures), if the consensus value is v, then some process invoked $x.\mathsf{propose}(v)$.

Strong consensus objects: [MMRT03] A strong (binary) consensus object x strengthens the second condition above to read: (2) If the consensus value is v, then some *correct* process invoked $x.\mathsf{propose}(v)$.[1]

In our protocols, we use both multi-writer and single-writer sticky bits. In the Byzantine setting, a single-writer sticky bit provides stronger properties than a single-writer one-bit register, in that a Byzantine writer cannot cause different processes to read different non-$\perp$-values from the same sticky bit. Notice also that a weak consensus object can be trivially implemented from a single sticky bit, and that in the crash-failure only case, weak consensus and strong consensus are identical.

2.3 Fault tolerance

It is shown in [MMRT03] that strong consensus objects tolerating t Byzantine failures do not exist when $n \leq 3t$, and do exist when $n \geq (2t+1)(t+1)$. In this paper, we close this gap, showing that strong consensus objects tolerating t Byzantine failures exist whenever $n \geq 3t+1$. (Technically, this means that a "strong consensus object" is one that provides the strong consensus property when n and t have the specified relationship, and has arbitrary behavior otherwise.)

As indicated in [MMRT03], strong consensus objects have inherently nonsequential runs: the additional condition, using redundancy to mask failures, requires that at least $t+1$ processes invoke $x.\mathsf{propose}()$ before any correct process returns from this operation. However, we require that if sufficiently many steps by correct processes are taken, then operations should complete. Specifically, for any operation $x.\mathsf{op}$, we say that:

- Operation $x.\mathsf{op}$ can *tolerate* t failures if $x.\mathsf{op}$, when executed by a correct process, eventually completes in any run ρ in which at least $n-t$ correct processes invoke $x.\mathsf{op}$.
- Operation $x.\mathsf{op}$ is *t-resilient* if $x.\mathsf{op}$, when executed by a correct process, eventually completes in any run in which each of at least $n-t$ correct processes infinitely often has a pending invocation of $x.\mathsf{op}$.

We next define fault-tolerant objects.

- Object o can *tolerate* t failures if all the operations o supports can *tolerate* t failures.
- Object o is *t-resilient* if all the operations o supports are t-resilient.

Notice that an object that can tolerate t failures is t-resilient, but not vice versa. In [MMRT03], an object that can *tolerate* t failures is called a t-threshold object. In these definitions, it may seem odd that termination is guaranteed only when correct processes access the object using the *same* operation. On the surface, it seems more natural to require termination in runs where at least $n-t$ correct processes access the object via *any* operation. Our definitions are actually more general, since one could encode different operations to be invocations of a single operation with different operands.

The t-resilience property, is appropriate for constructions in which each process requires the active participation of other processes in order to complete its operation—hence, it assures termination only when other (correct) processes continue to access the implemented object with the same operation. The property of *tolerating t failures* (the t-threshold property) is a stronger condition, requiring each operation by a correct process to terminate once at least $n-t$ correct processes have invoked that operation. This condition makes the most sense for "one-shot" objects, such as consensus or election.

The main positive result in [MMRT03] shows that there is a t-resilient universal construction out of wait-free sticky bits, in a Byzantine shared memory environment, when the number of failures t is limited. This leaves open the question of whether the same is true when t-resilient is replaced with t-threshold.

[1] Note that in the binary case, this definition of strong consensus coincides with one that only requires the consensus value to be v when all correct processes have the same input v. In the nonbinary case, this definition is strictly stronger.

As in [MMRT03], we use wait-free sticky bits to implement strong consensus objects that tolerate t Byzantine failures. These are in turn used to implement arbitrary t-resilient objects. (Throughout, atomic registers, sticky bits and any other primitive objects are assumed to be wait-free.)

3 Efficient strong consensus protocols

In this section, we present strong consensus protocols based on an idea of Berman and Garay [BG89, Mis89] for performing consensus in the message passing model. Their idea was to run a protocol in phases, where each phase preserves agreement, and if coordinated by a correct process, assures validity. Concatenating $t+1$ phases guarantees at least one is coordinated by a correct process.

We first show in Section 3.1 how to use sticky bits to implement a protocol phase with similar properties for the shared memory model. In Section 3.2, we show how several strong consensus protocols with different desirable properties can be constructed from these protocol phases. Our protocols are easily modifiable to implement nonbinary consensus, as shown in Section 3.3.

3.1 A protocol phase

A protocol phase makes use of two types of shared objects. First, a phase uses n *personal* sticky bits, $s_i : p_i \in P$, (where P is the set of all processes, $|P| = n$). Each s_i is writable only by the single process p_i. Second, a phase also uses a single sticky bit, S, which is writable by some set of $t+1$ processes. We call the $t+1$ processes in $\mathrm{ACL}(s_i.\mathsf{set}())$ *active* in the phase. Each process p_i enters the phase with a proposed consensus value in_i and leaves with the output value out_i.

Operation of a protocol phase, for each process $p_i \in P$:

1. Perform the wait-free $s_i.\mathsf{set}(in_i)$ operation. (That is, assign in_i to the personal sticky bit s_i.)
2. Perform wait-free reads of the personal sticky bits $s_1 \ldots s_n$ until seeing at least $t+1$ distinct occurrences of some value v other than $\perp$.
3. If p_i has write access to S, then perform the wait-free $S.\mathsf{set}(v)$ operation. (Of course, the first such scheduled process succeeds; the value of S does not change after that.)
4. Perform wait-free reads of S until returning a value v other than $\perp$. (Note that S could only return $\perp$ if p_i does not have write access to S.)
5. Perform wait-free reads of the personal sticky bits $s_1 \ldots, s_n$ until at least $n-t$ return with values other than $\perp$. If the value v read in step 4 occurs in at least $t+1$ of the values read, return $out_i = v$ (and say that p_i *supports* v in this phase), else return $out_i = \overline{v}$.

Lemma 3.1 *A protocol phase has the following properties (given $n \geq 3t+1$ and the bound t on the number of Byzantine processes):*

1. *If at least $n-t$ correct processes enter a phase, all correct processes eventually exit it.*
2. *The output value of any correct process is the input value of some correct process.*
3. *If all the active processes are correct, all correct processes exit the phase with the same value v.*

Proof: 1. The first and third steps are wait-free. Once $n-t$ correct processes finish the first step, since $n-t \geq 2t+1$, some value must occur at least $t+1$ times, and the second and fifth steps must eventually terminate. The fourth step of all processes must terminate once a single correct active process executes the third step, and this must eventually happen because there are $t+1$ active processes, so at least one of the active processes must be correct and will therefore perform step 3.

2. This follows because the output value of any correct process appears in at least $t+1$ personal sticky bits, so one must have been set by a correct process.

3. If all the active processes are correct, then the value v that S is set to will be supported by every correct process, and therefore every correct process with exit the phase with value v. ■

Now consider a consensus protocol, *Schema*, constructed by chaining finitely many separate protocol phases (with different sticky bits) together in a fixed sequence. Each process enters the first phase with its proposed value, uses the value returned from each phase as the input to the next phase, and returns as the protocol output the value returned from the last phase.

Lemma 3.2 *Given $n \geq 3t+1$ and the bound t on the number of Byzantine processes, if any phase has only correct active processes, Schema implements strong consensus.*

Proof: Part 1 of Lemma 3.1 guarantees the correct processes eventually exit each phase and so *Schema*. Part 2 guarantees the correct processes enter each phase with a valid input. The assumption and part 3 guarantee the correct processes eventually agree in a phase, and part 2 guarantees that they value they agree on is valid and that the correct processes do not change their value thereafter. ■

3.2 Strong consensus protocols

We present three protocols that use the protocol phases of Section 3.1 to implement strong consensus. The first two use different techniques for guaranteeing that some phase has only correct active processes, so Lemma 3.1 applies. The third replaces this requirement with a voting step at the end. The first protocol (Theorem 3.3) works for any $n \geq 3t+1$, but requires exponentially many potentially powerful objects. Since Malkhi et al. [MMRT03] show that $n \geq 3t+1$ is necessary for strong consensus in this model, our protocol is optimal in terms of the ratio between t and n. The second protocol (Theorem 3.5) uses only $t+1$ potentially powerful objects, but requires $n \geq (t+1)^2$. The third protocol (Theorem 3.6) is surprising because it works even if the faulty processes have access to all the potentially powerful objects. It modifies *Schema* by replacing the requirement that some phase contains only correct active processes by a voting step at the end. It uses only t potentially powerful objects and requires $n \geq t^2+5t+1$.

Theorem 3.3 *A strong consensus object tolerating t Byzantine failures can be implemented using $(t+1)$-*set()*, n-*read* sticky bits and 1-*set()*, n-*read* sticky bits, provided that $n \geq 3t+1$.*

Proof: Let P' be a subset of P with $2t+1$ processes. The protocol consists of $\binom{2t+1}{t}$ phases, each following the other, where the active processes in each phase consist of a distinct subset of $t+1$ processes from P'. Since only t processes are faulty, one such phase contains only correct active processes, and the theorem follows from Lemma 3.2. ■

One of the main results in [MMRT03] shows that there is a t-resilient universal construction out of wait-free sticky bits and strong consensus objects tolerating t Byzantine failures. Using this universality result and Theorem 3.3, we get that:

Corollary 3.4 *Any t-resilient object can be implemented using $(t+1)$-*set()*, n-*read* sticky bits and 1-*set()*, n-*read* sticky bits, provided that $n \geq 3t+1$, where t is the bound on the number of Byzantine failures.*

Though optimal in n and t, the protocol of Theorem 3.3 is not efficient in time or the number of potentially powerful objects, as it uses a number of rounds and of potentially powerful objects exponential in t. We show in Section 4 that the space bound is inherent: an exponential number of potentially powerful objects is *required* when $n = 3t+1$.

The following instantiation of *Schema* uses only $t+1$ rounds and $t+1$ potentially powerful objects, but requires $n \geq (t+1)^2$. In this instantiation, a completely new set of $t+1$ active processes is used for each protocol phase:

Theorem 3.5 *A strong consensus object that can tolerate t Byzantine failures can be implemented using $t+1$ $(t+1)$-set(), n-read sticky bits, together with 1-set(), n-read sticky bits, provided $n \geq (t+1)^2$.*

Proof: There are $t+1$ phases, each with a disjoint set of $t+1$ active processes, so for at least one phase all active processes are correct. The result follows by Lemma 3.2. ■

Theorems 3.3 and 3.5 are two extreme points in a tradeoff between the number of potentially powerful objects and the ratio of t to n. We examine this tradeoff more closely in Section 4.2. First, we present a final protocol that is surprising because there are only t potentially powerful objects, and therefore it works even if t Byzantine processes can access *all* the potentially powerful objects.

Theorem 3.6 *A strong consensus object tolerating t Byzantine failures can be implemented using t potentially powerful $(t+1)$-set(), n-read sticky bits, together with 1-set(), n-read sticky bits, provided that $n \geq t^2 + 5t + 1$.*

Proof: We modify the protocol of Theorem 3.5 by omitting the last phase. That is, there are t phases, each involving a disjoint set of $t+1$ active processes accessing the $(t+1)$-set(), n-read sticky bit for that phase. We then designate exactly $4t+1$ additional processes (there are at least that many) not active in any phase as *voters*. Note that either some phase contains only correct active processes, or all the voters are correct. Each of the voting processes takes its output from the last phase and writes its resulting value in a personal sticky bit. After the last phase, all processes read the voters' personal sticky bits, and decide on the first value they see occurring $2t+1$ times.

To see that this works, first note that at most one value can occur $2t+1$ or more times among the $4t+1$ voters. We now show one value must occur at least that often, and that the value is valid. By parts 2 and 3 of Lemma 3.1, if there is a phase in which all the active process are correct, then all correct voters will write the same valid value to their own personal sticky bit. Since in this case at least $3t+1$ voters are correct, eventually at least $3t+1 \geq 2t+1$ votes will be written and will agree on some valid value v. If there is no phase in which all the active processes are correct, then as argued above, no voter is faulty. In this case, all voters will write a valid value, so one value will be written at least $\lceil (4t+1)/2 \rceil = 2t+1$ times. ■

Since there are only t potentially powerful objects, this algorithm works even in the case that no single potentially powerful object is accessible exclusively by correct processes. In this case, the size, $t+1$, of the ACLs is needed to ensure that each potentially powerful object will eventually be written (and so other processes can wait for it to be set), rather than ensuring that some potentially powerful object will be written by a correct process. Other protocols derived from the protocol *Schema*, such as the algorithm in the proof of Theorem 3.3, can be similarly modified, removing one protocol phase and replacing it with a set of $4t+1$ voters.

One might expect that such tricks could be used to reduce the number of potentially powerful objects even further. For example, can strong consensus be implemented with even fewer than t potentially powerful sticky bits? Do t sticky bits suffice for $n = 3t+1$? (The t-bit algorithm of Theorem 3.6 requires $n \geq t^2+5t+1$.) We explore these questions in Section 4.

3.3 Implementing Strong k-consensus

In this section, we describe how our protocols can extend to k-valued strong consensus.

Strong k-consensus objects: A strong k-consensus object x is an object with one operation: x.propose(v), where $v \in \{0, \ldots, k-1\}$, satisfying: (1) In any run, the x.propose() operation returns the same value, called the *consensus value*, to every correct process that invokes it. (2) If the consensus value is v, then some *correct* process invoked x.propose(v).

To implement k-consensus, we modify the last step of a protocol phase from Section 3.1 to return $out_i = in_i$ in the case that the value v read in step 4 is not supported by p_i in this phase. (This ensures that

the output value of any correct process is always the input value of some correct process.) In order to ensure that step 3 always terminates, it becomes necessary to have $n \geq (k+1)t+1$. This modification yields the corresponding results; the straightforward modifications to the proofs are omitted.

Lemma 3.7 *A modified protocol phase has the following properties (given $n \geq (k+1)t+1$ and the bound t on the number of Byzantine processes):*

1. *If at least $n-t$ correct processes enter a phase, all correct processes eventually exit it.*
2. *The output value of any correct process is the input value of some correct process.*
3. *If all the active processes are correct, all correct processes exit the phase with the same value v.*

Theorem 3.8 *A strong k-consensus object tolerating t Byzantine failures can be implemented using $(t+1)$-*set()*, n-*read* *sticky bits and* 1-set(), n-read *sticky bits, provided that $n \geq (k+1)t+1$.*

Theorem 3.9 *A strong k-consensus object that can tolerate t Byzantine failures can be implemented using $t+1$ $(t+1)$-*set()*, n-*read* *sticky bits, together with* 1-set(), n-read *sticky bits, provided $n \geq \max((t+1)^2, (k+1)t+1)$.*

4 Lower Bounds and Tradeoffs

The protocols of Theorem 3.3 and Theorem 3.5 represent different points on a tradeoff between efficiency and usefulness for more values of n. The protocol of Theorem 3.3 achieves strong consensus whenever $n \geq 3t+1$, but requires an exponential number (in t) of potentially powerful sticky bits. In contrast, the protocol of Theorem 3.5 uses only a polynomial number of potentially powerful sticky bits, but is only guaranteed to achieve strong consensus when $n \geq (t+1)^2$.

This raises the question of whether these results are the best possible. For example, can we do with much fewer (e.g., polynomially many) potentially powerful objects and still achieve strong consensus whenever $n \geq 3t+1$?

In this section, we show that the answer to this question is no. More generally, we show tight asymptotic tradeoff between the number, k, of potentially powerful sticky bits or a more general class of potentially powerful objects that must be used and the number of processes, n (as a function of the number of possible failures t) in order to achieve strong consensus. This tradeoff is essentially given by $k = t \cdot 2^{\Theta(t^2/n)}$. In particular, when $n = 3t+1$, an exponential number of such objects is indeed necessary. Interestingly, we also obtain a protocol for $n = O(t^2/\log t)$ that uses a polynomial number of sticky bits. Such a protocol was only previously known for $n = \Omega(t^2)$ [MMRT03].

We begin by showing in Theorem 4.1 that the upper bound t on the number of faulty processes is in fact a lower bound, regardless of the number n of processes, on the number of potentially powerful sticky bits needed for strong consensus. In the ensuing subsections, we examine the general tradeoff between the number of sticky bits or related objects and the size of n as a function of t for implementing strong consensus. The reader may wish to review the definitions in Section 2 before proceeding.

Theorem 4.1 *No t-resilient strong consensus object can be implemented from fewer than t potentially powerful sticky bits (using no other potentially powerful objects), where t is the bound on the number of Byzantine failures.*

Proof: Suppose such a protocol exists. Since it uses at most $t-1$ potentially powerful sticky bits, the protocol must remain 1-resilient even in the case that $t-1$ Byzantine processes first set these bits to 0 before any correct process takes a step, and subsequently take no action. These sticky bits are clearly useless: omitting them and the $t-1$ Byzantine processes results in a 1-resilient protocol with *no* potentially powerful objects that uses registers and single-writer sticky bits. Obviously, this protocol is also correct in the crash-fault model (against the failure of a single process). But in the crash model, single-writer sticky bits can be implemented by registers, and the resulting protocol contradicts the well-known results of Fischer, Lynch and Paterson [FLP85] and Loui and Abu-Amara [LA87]. ■

4.1 The ACL Theorem

We next prove a general theorem establishing a necessary condition for implementing *weak* consensus even in the presence of only *crash* failures. Dwork, Herlihy, and Waarts [DHW97] show that any implementation of wait-free n-process consensus algorithm includes an object, say o, that can be accessed by n processes. Our result generalizes theirs in two ways: it shows that the object o, must be potentially powerful; and it covers also implementations that tolerate $t < n$ failures (not just wait-free implementations). We show that when at most t out of n processors may crash, the weak consensus problem is only solvable in shared memory systems containing a potentially powerful object o such that $\mathrm{ACL}_{\mathsf{pow}}(o) \geq t+1$; our proof is similar to proofs of [FLP85, LA87]. Note that it is possible to implement weak consensus tolerating up to t Byzantine failures using a single potentially powerful object o with $\mathrm{ACL}_{\mathsf{pow}}(o) \geq t+1$, such as a weak consensus object o defined with $|\mathrm{ACL}(o.\mathsf{propose}())| = t+1$ or a sticky bit o with $|\mathrm{ACL}(o.\mathsf{set}())| = t+1$. Hence, Theorem 4.2 is tight in some sense.

Theorem 4.2 *Any weak consensus protocol that tolerates $t \geq 1$ crash failures must use at least one object, o, such that $|ACL_{\mathsf{pow}}(o)| \geq t+1$.*

Proof: We denote a run of a protocol by the sequence in which processes invoke operations. A finite run x is *v-valent* if in all extensions of x where a decision is made, the decision value is v ($v \in \{0,1\}$). A run is *univalent* if it is either 0-valent or 1-valent, otherwise it is *bivalent*. In the following, P denotes a set of processes, x and x' denote runs and $x'p$ is an extension of the run x' by one step of process p.

Assume π is a consensus protocol that can tolerate t crash failures. By familiar arguments, π has an empty bivalent run x_0. We begin with x_0 and pursue the following round-robin *bivalence-preserving scheduling* discipline:

$x := x_0$; $P := \emptyset$; $i := 0$;
`repeat`
  $j := i+1$
  `if` x has a bivalent extension $x'p_j$
  `then` $x := x'p_j$
  `else` $P := P \cup \{p_j\}$
  $i := (i+1) \bmod n$
`until` $|P| = t+1$.

If this procedure does not terminate, then there is an infinite run with only bivalent finite prefixes in which $n-t$ processes are correct. However, the existence of such a run contradicts the definition of consensus protocols that can tolerate t crash failures. Hence, the procedure will terminate with some bivalent finite run x, and a set P of $t+1$ processes such that any extension $x'p$ of x, for any process p in P, is univalent.

Pick any $p \in P$, and let v be such that the run xp is v-valent. Since x is bivalent, there is a shortest extension z of x which is $\overline{v}$-valent. (See Figure 1(a).)

Let z' be the longest prefix of z that does not contain any step of p, and note that either $z = z'$ or $z = z'p$. From the assumption about z', it follows that $z'p$ is $\overline{v}$-valent, and $z' \neq x$. (See Figure 1(b).)

Consider the extensions of x that are also prefixes of z'. Since xp and $z'p$ have opposite valence, there must exist an extension y of x and a process $q \neq p$, $x \leq y < yq \leq z'$, such that yp and yqp are univalent but with opposite valence. Hence y is bivalent, and it further follows that y is a P-free extension of x. (See Figure 1(c).)

Familiar case analyses [FLP85, LA87] preclude p from invoking wait-free read or write() operations at y: If p's next step is a read operation, then ypq and yqp are indistinguishable to processes other than p, yet their p-free extensions must have opposite valence, a contradiction. If p's next step is a write() operation, then yp and yqp are indistinguishable to processes other than q, yet their q-free extensions must have opposite valence, a contradiction. Thus we conclude that the next operation of p at y is potentially powerful, and as y is a p-free extension of x, it follows that the next operation of p at x is also potentially powerful (since they are the same operation). Since we chose p arbitrarily from P, the identical argument for each member

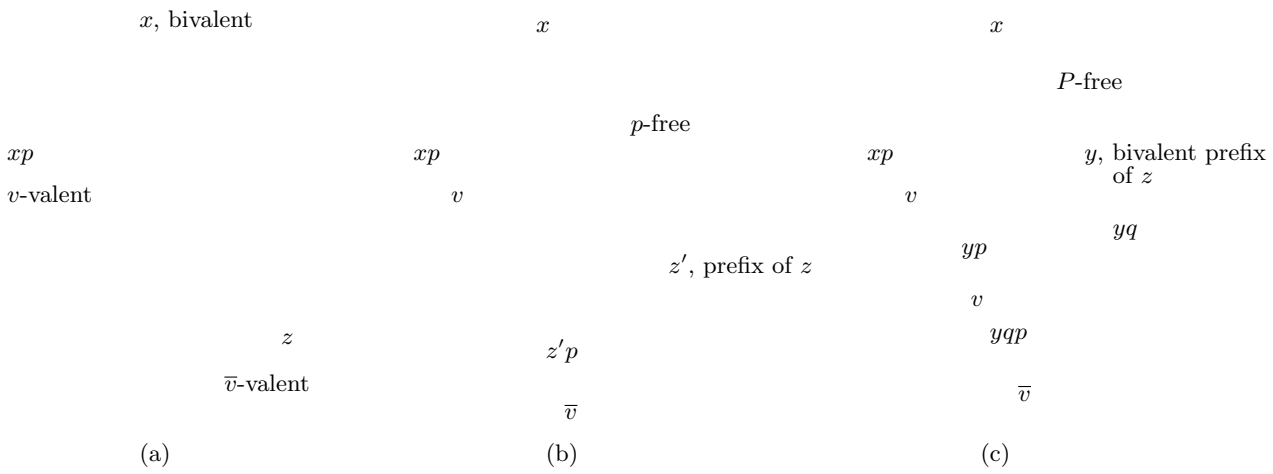

Figure 1: Illustration of runs in proof of Theorem 4.2

of P implies that the the next operation of *any* member of P at x is potentially powerful, and hence the next operation of *any* member of P at the P-free extension y of x is potentially powerful.

It remains to show that the next operations of *all* the members of P at y access the same object. Let o be the object accessed by p in the last step of yp. If q does not access o in the last step of yq, then ypq and yqp are indistinguishable, a contradiction. Now consider any $p' \in P$, and the single-step extension yp' of y. If p' accesses an object other than o in the last step of yp', then note first that p' is neither p nor q. Then either yp' is v-valent, and the indistinguishability of $yp'qp$ and $yqpp'$ leads to a contradiction, or yp' is $\overline{v}$-valent, and similarly the indistinguishability of ypp' and $yp'p$ leads to a contradiction. It follows that all $t+1$ processes in P invoke potentially powerful operations on o. We conclude that $|\mathrm{ACL}_{\mathsf{pow}}(o)| \geq t+1$. ■

The famous impossibility result from [FLP85, LA87] follows immediately from Theorem 4.2.

Corollary 4.3 *There is no weak consensus protocol tolerating even one crash failure that uses only atomic read/write registers.*

Proof: By Theorem 4.2, any weak consensus protocol that tolerates one crash failures must use at least one object, o, such that $|\mathrm{ACL}_{\mathsf{pow}}(o)| \geq 2$. However, for any atomic register r, $|\mathrm{ACL}_{\mathsf{pow}}(r)| = 0$. ■

As we will see, Theorem 4.2, in combination with the protocols of the previous section, is a powerful tool in establishing an asymptotic tradeoff on the number of sticky bits (or other potentially powerful objects) necessary to implement strong consensus, as n varies relative to t.

4.2 Subvertible Objects and Immunity

In this and the following subsections, we investigate a tradeoff between the number of potentially powerful sticky bits and the number n of processes (as a function of the number t of possible failures) that must be used in order to achieve strong consensus. Since a strong consensus object s would of course trivially implement itself (with $\mathrm{ACL}_{\mathsf{pow}}(s.\mathsf{propose}()) = n$), some care is needed to generalize the question from the specific case of how many sticky bits are necessary to how many objects are necessary in order to implement strong consensus. Noting that the relevant property of sticky bits is that a single Byzantine process with access to a sticky bit can render it useless to the correct processes, we generalize from the notion of sticky bits to "subvertible" objects.

Accordingly, we define an object o to be *subvertible* if it is potentially powerful and any Byzantine process in ACL(o) can cause operations by correct processes to be useless—that is, if there are runs in which the return values of correct processes are dependent only on the operations of the Byzantine process.[2] Sticky bits are subvertible, since a Byzantine process can invoke $\mathsf{set}(0)$ before any correct process takes a step. Obviously, if the object supports $\mathsf{write}()$ operations it is subvertible, and of course, strong consensus itself is not subvertible.

This section uses a combinatorial analysis to provide a tight asymptotic tradeoff between the number, k, of subvertible objects that must be used and the number n of processes (as a function of the number t of possible failures), in the case that only subvertible objects are used.

Correctness of the first two protocols in Section 3 depends on there being enough such objects that at least one is accessed *only* by correct processes. This property is a combinatorial property of the sticky bit access control lists, which we call being "t-immune": no set of t (faulty) processes can subvert all the potentially powerful sticky bits. This is formalized in the following definition. We denote the set $\{1, 2, \ldots, n\}$ by $[n]$. A collection of subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ of the domain $[n]$ is *m-immune* if for every set $T \subseteq [n]$ of m elements there exists $S_j \in \mathcal{S}$ such that $T \cap S_j = \emptyset$. We note that related objects have been studied extensively in the past [Fur91, GGL95].

Theorem 3.6 shows that strong consensus protocols can exist even when the access lists of the potentially powerful objects are not t-immune. However, we now argue that m-immunity (for some m that depends on t) is a necessary condition. As a simple corollary of Theorem 4.2 (and generalizing Theorem 4.1) we have the following:

Corollary 4.4 *Let π be any strong consensus protocol that tolerates $t_1 \geq 1$ crash failures and $t_2 \geq 1$ Byzantine failures and uses only subvertible objects. Let $\mathcal{S}$ be the collection of $ACL_{\mathsf{pow}}(o)$ for all potentially powerful objects o used by π that are of size at least $t_1 + 1$. Then $\mathcal{S}$ is t_2-immune.*

Proof: Assume to the contrary that $T_2 = \{i_1, \ldots, i_{t_2}\}$ is a set of t_2 processes that cover $\mathcal{S}$ (i.e., for all $S_j \in \mathcal{S}$, $T_2 \cap S_j \neq \emptyset$). Then π contains runs in which all the processes in T_2 are Byzantine. These processes can subvert all the objects with large (size $t_1 + 1$) access lists, making these objects useless in fighting the remaining t_1 crash failures. More formally, it is easy to modify π so that it still tolerates t_1 crash failures but uses no object o with $\mathrm{ACL}_{\mathsf{pow}}(o)$ bigger than t_1. Since this is a contradiction to Theorem 4.2, the corollary follows. ■

Noting that Byzantine failures can choose to simply crash, the next corollary follows trivially from Corollary 4.4 by taking $t_1 = \lfloor (1-\alpha)t \rfloor$ and $t_2 = \lceil \alpha t \rceil$.

Corollary 4.5 *Let π be any strong consensus protocol that tolerates t Byzantine failures and uses only subvertible objects. Let $0 < \alpha < 1$ be some constant such that $(1-\alpha)t \geq 1$, and let $\mathcal{S}$ be the collection of $ACL_{\mathsf{pow}}(o)$ for all potentially powerful objects o used by π that are of size at least $\lfloor (1-\alpha)t \rfloor + 1$. Then $\mathcal{S}$ is $\lceil \alpha t \rceil$-immune.*

4.3 Bounds on the Size of m-Immune Collections

In light of Corollary 4.5 and the proof of Theorem 3.3, both upper and lower bounds on the number of subvertible objects needed by Byzantine consensus protocols (Theorems 4.8 and 4.9) can be derived from corresponding bounds on the number of sets in m-immune collections. Such bounds are given in this section. The minimum possible number of subsets in an m-immune collection of subsets of cardinality at least $t+1$ each in the domain $[n]$ is precisely the minimum number of edges in a $t+1$-uniform hypergraph on n vertices that contains no independent set of size $n-m$. This number is the hypergraph Turán number $T(n, n-m, t+1)$, in the notation of [Fur91]. Although these numbers have been investigated extensively, the existing results focus on the cases of fixed values of $t+1$ (the size of each edge) and $n-m$ (the forbidden

[2] A formal definition of subvertible objects would require considerable machinery and is beyond the scope of this paper. A concerned reader may substitute "potentially powerful sticky bits" in place of "subvertible objects" in the remainder of the paper.

size of a maximum independent set), and large n, whereas in our case the parameters are different. We were therefore unable to deduce our results from previous work, but apply techniques similar to some of the existing ones to derive our bounds (cf. [Fur91, GGL95]). For simplicity, we concentrate in the next theorem on the case $n \geq 3t+1$ which is the interesting one for our application.

Theorem 4.6 *Let $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ be an m-immune collection of subsets of the domain $[n]$. If each set $S_j \in \mathcal{S}$ contains at least $t+1$ elements, where $n \geq 3t+1$, then $n \geq t+m+1$, and*

$$k \geq \max\{m+1, m \cdot 2^{\Omega(t \cdot m/n)}\}.$$

Proof: That $k \geq m+1$ is trivial (any collection of less than $m+1$ subsets can be covered by m elements). Similarly, n must be at least $t+m+1$ as otherwise any subset of $[n]$ of cardinality m intersects all subsets of cardinality at least $t+1$ (which includes all of the sets in $\mathcal{S}$). It remains to prove that $k \geq m \cdot 2^{\Omega(t \cdot m/n)}$, for which we apply the probabilistic method (see [AS00]). We can assume without loss of generality that $t \cdot m/n = \Omega(1)$ (as otherwise this inequality follows from $k \geq m+1$). Also assume without loss of generality that m is even. Consider a subset $T' \subset [n]$ obtained by choosing $m/2$ elements randomly with replacement from $[n]$. Then $|T'| \leq m/2$, and for any $j \in [k]$,

$$\Pr[T' \cap S_j = \emptyset] \leq \left(1 - \frac{t+1}{n}\right)^{m/2}.$$

Therefore, the expected number of $j \in [k]$ such that $T' \cap S_j = \emptyset$ is at most $k\left(1 - \frac{t+1}{n}\right)^{m/2}$. If $k\left(1 - \frac{t+1}{n}\right)^{m/2} \leq m/2$ then there exists an assignment to the set T' that covers all but $m/2$ of the subsets S_j. The remaining subsets can be covered by additional $m/2$ elements in $[n]$, which contradicts the assumption that $\mathcal{S}$ is m-immune. We can therefore conclude that $k \geq m/2 \cdot \left(1 - \frac{t+1}{n}\right)^{-m/2}$. Since $n \geq 3t+1$ and we also have $n \geq t+m+1$, and $m > 0$ (otherwise this part of the proof is trivial), we get that $\frac{t+1}{n} \leq 1/2$. Therefore, $\left(1 - \frac{t+1}{n}\right)^{-1} = e^{\Omega((t+1)/n)}$ (by the Taylor expansion of e^{-x}). Since we assumed $t \cdot m/n = \Omega(1)$ we can conclude that $m/2 \cdot \left(1 - \frac{t+1}{n}\right)^{-m/2} = m \cdot 2^{\Omega(t \cdot m/n)}$, completing the proof. ■

An upper bound on the number of sets in m-immune collections can also be obtained by an application of the probabilistic method. Nevertheless, we are interested in an *explicit construction* of the m-immune collection (so that the resulting consensus protocol is explicit as well). The proof of the following theorem provides such a construction.

Theorem 4.7 *For any n, t and m with $n \geq t+m+1$, there exists an m-immune collection $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ of subsets of the domain $[n]$, such that (1) all the sets $S_j \in \mathcal{S}$ contain at least $t+1$ elements and (2)*

$$k \leq \max\{m+1, m \cdot 2^{O(t \cdot m/n)}\}.$$

Proof: First we consider two values of n for which a simple construction of the desired collection $\mathcal{S}$ exists: (a) If $n \geq (m+1)(t+1)$ we can simply define $\mathcal{S}$ to be a collection of $m+1$ disjoint subsets of size $t+1$. (b) If $n < 8(t+m)+1$, we can define $\mathcal{S}$ to contain all the subsets of size $t+1$ of the first $t+m+1$ elements. When $n < 8(t+m)+1$ this construction gives $k = \binom{t+m+1}{m} < 2^{t+m+1} = 2^{O(t \cdot m/n)}$ as claimed. We can therefore assume without loss of generality that $n < (m+1)(t+1)$. We can also assume without loss of generality that $n, t+1$ and m are all (positive) powers of 2 and $n \geq 2(m+t)+1$. To justify this assumption, we note that it is sufficient to construct our collection for $n' \leq n$, $t' \geq t$, and $m' \geq m$ that are the closest values such that $n', t'+1$ and m' are all powers of 2. As long as $n \geq 8(t+m)+1$, we have that $n' \geq 2(t'+m')+1$, (which in particular guarantees that $n' \geq t'+m'+1$). As argued above, it is safe to assume that $n \geq 8(t+m)+1$.

We can now define the desired m-immune collection. Set $\ell = 2(t+1)m/n$. By our assumptions ℓ is a positive integer (as both $2(t+1)m$ and n are powers of 2 and $n \leq 2(t+1)m$). Set $r = n/2(t+1)$. By our assumptions this also is an integer and furthermore $r < m$. Now, let us divide the set $[n]$ into $2\ell r = 2m$ disjoint subsets of equal size: $A_{1,1}, \ldots A_{1,2\ell}, A_{2,1}, \ldots, A_{2,2\ell} \ldots, A_{r,2\ell}$ (this is again possible since n and m are

powers of 2 and $n > 2m$). We can now define the collection $\mathcal{S}$. For any $i \in [r]$ and any ℓ distinct indices $j_1, j_2, \ldots, j_\ell$ in $[2\ell]$, the collection contains the set

$$S_{i,j_1,j_2,\ldots,j_\ell} = A_{i,j_1} \cup A_{i,j_2} \cup \ldots \cup A_{i,j_\ell}.$$

By definition, the number of sets in the collection is $r \cdot \binom{2\ell}{\ell} < r \cdot 2^{2\ell} = m \cdot 2^{O(t \cdot m/n)}$. Furthermore, each set contains $n/2r = t+1$ elements. It remains to show that $\mathcal{S}$ is m-immune. Let T be any subset of $[n]$ of size m. By averaging, there exists at least one $i \in [r]$ such that T contains at most $m/r = \ell$ elements in $\cup_{j=1}^{2\ell} A_{i,j}$. Therefore, T contains elements in at most ℓ sets $\{A_{i,j}\}_{j=1}^{2\ell}$. Therefore, there exist distinct indices $j_1, j_2, \ldots, j_\ell$ in $[2\ell]$, such that $T \cap S_{i,j_1,j_2,\ldots,j_\ell} = \emptyset$. ■

4.4 The Tradeoff Results

Putting it all together, we have the following complementary theorems.

Theorem 4.8 *For any $n \geq 3t+1$, there exists a strong consensus protocol that tolerates t Byzantine failures and the only potentially powerful objects it uses are $\max\{t+1, t \cdot 2^{O(t^2/n)}\}$ sticky bits with access lists of size $t+1$.*

Proof: If $t = 0$, then strong consensus can be trivially achieved using one single-writer sticky bit. Otherwise, $t \geq 1$. Applying Theorem 4.7 with $m = t$ yields a t-immune collection of $\max\{t+1, t \cdot 2^{O(t^2/n)}\}$ subsets of $[n]$, each of size $t+1$. Taking those subsets as the access control lists of $\max\{t+1, t \cdot 2^{O(t^2/n)}\}$ sticky bits yields the desired protocol. ■

Theorem 4.9 *For any $n \geq 3t+1$ and any constant $0 < \alpha < 1$, any strong consensus protocol that tolerates t Byzantine failures using only subvertible objects must use at least $t \cdot 2^{\Omega(t^2/n)}$ subvertible objects with $|ACL_{\mathsf{pow}}(o)| \geq (1-\alpha)t$ for each such object o.*

Proof: If $(1-\alpha)t \geq 1$, then applying Corollary 4.5 we have that the collection of $\mathrm{ACL}_{\mathsf{pow}}(o)$ for all potentially powerful objects o used by the strong consensus protocol that are of size at least $\lfloor (1-\alpha)t \rfloor + 1 \geq (1-\alpha)t$ is $\lceil \alpha t \rceil$-immune. The theorem now follows from Theorem 4.6. In case $(1-\alpha)t < 1$, then in particular t is also a constant, and the theorem follows as above by applying Corollary 4.5 (and then Theorem 4.6) with $\alpha' = 1 - 1/t$. ■

As a corollary of Theorem 4.8, we get the promised protocol for $n = O(t^2/\log t)$ that only uses a polynomial number of sticky bits.

Corollary 4.10 *For any $n \geq 3t+1$ such that $n = O(t^2/\log t)$, there exists a strong consensus protocol that tolerates t Byzantine failures and the only potentially powerful objects it uses are a polynomial number (in t) of sticky bits with access lists of size $t+1$.*

5 Conclusions

We presented a strong consensus protocol that can tolerate t Byzantine failures (and therefore a universality result for linearizable t-resilient objects) from wait-free sticky bits for $n \geq 3t+1$. We demonstrated a tight tradeoff between the fault tolerance and the efficiency of any strong consensus protocol using subvertible objects such as sticky bits, in particular showing that any strong consensus protocol that works for all $n \geq 3t$ and uses only subvertible objects must use an exponential number of objects. The tradeoff also implies a polynomial time strong consensus protocol that can tolerate t Byzantine failures for $n = O(t^2/\log t)$.

It remains open whether sticky bits are universal for $n \leq 3t$. Since strong consensus is not possible for $n \leq 3t$, a different universality construction not going through strong consensus would be needed. (The impossibility of strong consensus for $n \leq 3t$ rests on the logical inconsistency of the object specification, and does not reflect the implementability of other, especially linearizable, object types.)

Acknowledgments

We would like to thank Yuval Ishai and Juan Garay for useful discussions. We also thank the anonymous reviewers for their extensive and helpful comments.

References

[AGMT95] Y. Afek, D. Greenberg, M. Merritt, and G. Taubenfeld. Computing with faulty shared memory. *Journal of the ACM*, 42(6):1231–1274, November 1995.

[AS00] N. Alon and J. Spencer, *The Probabilistic Method*, Second Edition, Wiley, New York, 2000.

[Att00] P.C. Attie. Wait-free Byzantine Agreement. Technical Report NU-CCS-00-02, College of Computer Science, Northeastern University, May 2000.

[Boe04] H-J. Boehm. An almost non-blocking stack. In *Proc. 23rd ACM Symp. on Principles of Distributed Computing*, pages 40–49, July 2004.

[CL99] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation – OSDI'99*, February, 1999, New Orleans, LA.

[BG89] P. Berman and J.A. Garay. Asymptotical optimal distributed consensus. *Proceedings of the 16th International Colloquium on Automata, Languages and Programming (ICALP 89)*, LNCS 372, pages 80–94, 1989.

[DHLM04] S. Doherty, M. Herlihy, V. Luchangco, and M. Moir. Bringing practical lock-free synchronization to 64-bit applications. In *Proc. 23rd ACM Symp. on Principles of Distributed Computing*, pages 31–39, July 2004.

[DHW97] C. Dwork, M. Herlihy, and O. Waarts. Contention in shared memory algorithms. *J. ACM*, 44(6):779–805, November 1997.

[FHS98] F. Fich, M. Herlihy, and N. Shavit, On the Space Complexity of Randomized Synchronization. *Journal of the ACM*, 45(5):843–862, September 1998.

[FLP85] M.J. Fischer, N.A. Lynch, and M.S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[Fur91] Z. Füredi, Turán type problems. Surveys in combinatorics, 1991 (Guildford, 1991), 253–300, London Math. Soc. Lecture Note Ser., 166, Cambridge Univ. Press, Cambridge, 1991.

[GGL95] M. Grötschel, R. L. Graham, and L. Lovász, *Handbook of Combinatorics.* Vol. 2, Chapter 24. MIT Press. 1995.

[HW90] M.P. Herlihy and J.M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems* 12(3):463–492, July 1990.

[Her91] M.P. Herlihy. Wait-free synchronization, *ACM Transactions on Programming Languages and Systems* 13(1):124–149, January 1991. A preliminary version appeared in PODC'88.

[JCT98] P. Jayanti, T. Chandra, and S. Toueg. Fault-tolerant wait-free shared objects. *Journal of the ACM*, 45(3):451–500, May 1998.

[JT92] P. Jayanti and S. Toueg. Some results on the impossibility, universality, and decidability of consensus. *Proc. of the 6th Int. Workshop on Distributed Algorithms* LNCS 647, pages 69–84, 1992.

[KMM98] K. P. Kihlstrom, L. E. Moser and P. M. Melliar-Smith. The SecureRing protocols for securing group communication. In *Proceedings of the 31st IEEE Hawaii Int. Conf. on System Sciences*, pages 317–326, January 1998.

[MS04] E. Ladan-Mozes and N. Shavit. An Optimistic Approach to Lock-Free FIFO Queues. In *Proceedings of the 18th International Symposium on Distributed Computing*, LNCS 3274, pages 117–131, 2004.

[Lea04] D. Lea. The Java concurrency package JSR-166. `http://gee.cs.oswego.edu/dl/concurrency-interest/index.html`.

[LA87] M.C. Loui and H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4:163–183, 1987.

[MMRT03] D. Malkhi, M. Merritt, M. Reiter, and G. Taubenfeld. Objects shared by Byzantine processes. *Distributed Computing* 16(1):37–48, 2003. A preliminary version appeared in *Proceedings of the 14th International Symposium on Distributed Computing (DISC 2000)*, LNCS 1914, pages 345–359, 2000.

[MR00] D. Malkhi and M. K. Reiter. An architecture for survivable coordination in large distributed systems. *IEEE Transactions on Knowledge and Data Engineering* 12(2):187–202, March/April 2000.

[MRTW02] M. Merritt, O. Reingold, G. Taubenfeld, and R. N. Wright, In *Proceedings of the 16th International Symposium on Distributed Computing (DISC 2002)*, LNCS 2508, pages 222–236, 2002.

[Mic04] M. M. Michael. Practical Lock-Free and Wait-Free LL/SC/VL Implementations Using 64-Bit CAS. In *18th International Symposium on Distributed Computing*, LNCS 3274, pages 144–158, 2004.

[Mis89] J. Misra. A simple proof of a simple consensus algorithm. *Information Processing Letters*, 33(1):21–24, 1989.

[PG89] F. M. Pittelli and H. Garcia-Molina. Reliable scheduling in a TMR database system. *ACM Transactions on Computer Systems*, 7(1):25–60, February 1989.

[Plo89] S.A. Plotkin. Sticky bits and universality of consensus. In *Proc. 8th ACM Symp. on Principles of Distributed Computing*, pages 159–175, August 1989.

[Rei96] M. K. Reiter. Distributing trust with the Rampart toolkit. *Communications of the ACM* 39(4):71–74, April 1996.

[SE+92] S. K. Shrivastava, P. D. Ezhilchelvan, N. A. Speirs, S. Tao, and A. Tully. Principal features of the VOLTAN family of reliable node architectures for distributed systems. *IEEE Trans. on Computers*, 41(5):542–549, May 1992.