Bosco: One-Step Byzantine Asynchronous Consensus*

Yee Jiun Song and Robbert van Renesse

Cornell University, Ithaca, NY 14850, USA

Abstract. Asynchronous Byzantine consensus algorithms are an important primitive for building Byzantine fault-tolerant systems. Algorithms for Byzantine consensus typically require at least two communication steps for decision; in many systems, this imposes a significant performance overhead. In this paper, we show that it is possible to design Byzantine fault-tolerant consensus algorithms that decide in one message latency under contention-free scenarios and still provide strong consistency guarantees when contention occurs. We define two variants of one-step asynchronous Byzantine consensus and show a lower bound on the number of processors needed for each. We present a Byzantine consensus algorithm, Bosco, for asynchronous networks that meets these bounds, even in the face of a strong network adversary.

1 Introduction

Informally, the consensus problem is the task of getting a set of processors to agree on a common value. This simple primitive can be used to implement atomic broadcast, replicated state machines, and view synchrony, thus making consensus an important building block in distributed systems.

Many variants of the consensus problem have been proposed. The differences between them lie mainly in the failure assumptions and the synchronicity assumptions. In this paper, we are concerned with Byzantine consensus in an asynchronous environment, i.e., faulty processors can behave in an arbitrary manner and there are no assumptions about the relative speed of processors nor about the timely delivery of messages.

Consensus algorithms allow processors to converge on a value by exchanging messages. Previous results have shown that algorithms that solve asynchronous Byzantine consensus must have correct executions that require at least two communication steps even in the absence of faults [1], where a single communication step is defined as a period of time where each processor can i) send messages; ii) receive messages; and iii) do local computations, in that order. However, this

* The authors were supported by AFRL award FA8750-06-2-0060 (CASTOR), NSF award 0424422 (TRUST), AFOSR award FA9550-06-1-0244 (AF-TRUST), DHS award 2006-CS-001-000001 (I3P), as well as by ISF, ISOC, CCR, and Intel Corporation. The views and conclusions herein are those of the authors.

G. Taubenfeld (Ed.): DISC 2008, LNCS 5218, pp. 438–450, 2008.

does not mean that such algorithms must *always* take two or more communication steps. We show that when there is no contention, it is possible for processors to decide a value in one communication step.

One-step decisions can improve performance for applications where contention is rare. Consider a replicated state machine: if a client broadcasts its operation to all machines, and there is no contention with other clients, then all correct machines propose the same operation and can respond to the client immediately. Thus an operation completes in just two message latencies, the same as for a Remote Procedure Call to an unreplicated service.

Previously such *one-step* asynchronous consensus algorithms have been proposed for crash failure assumptions [2,3,4,5,6,7]; Friedman et al. proposed a common coin-based one-step consensus algorithm that tolerates Byzantine failures and terminates with probability 1 but requires that the network scheduler has no knowledge of the common coin oracle [8]. In this paper, we consider one-step algorithms for Byzantine asynchronous consensus in the presence of a strong network adversary. We define two different notions of one-step Byzantine asynchronous algorithms and prove a lower bound for the number of processors that are required for each. Next we show that the lower bounds are tight by extending the work presented in [2] to handle Byzantine failures, resulting in Bosco, an algorithm that meets these bounds.

The rest of the paper is organized as follows: Section 2 defines the model and the Byzantine consensus problem; Section 3 proves lower bounds for the two versions of one-step Byzantine consensus; Section 4 describes Bosco, a one-step consensus algorithm; Section 5 discusses some properties of Bosco; Section 6 presents a brief survey of some related work; finally, Section 7 concludes.

2 The Byzantine Consensus Problem

The Byzantine consensus problem was first posed in [9], albeit for a synchronous environment. In this paper we focus on an asynchronous environment.

In this problem, there is a set of n processors $P = \{p, q, ...\}$ each of which have an initial value, 0 or 1. An unknown subset T of P contains *faulty* processors. These faulty processors may exhibit arbitrary (*aka* Byzantine) behavior, and may collude maliciously. Processors in $P - T$ are *correct* and behave according to some protocol. Processors communicate with each other by sending messages via a network. The network is assumed to be fully asynchronous but reliable, that is, messages may be arbitrarily delayed but between two correct processors, will be eventually be delivered. Links between processors are private so a Byzantine processor cannot forge a message from a correct processor.

In addition, we assume a *strong network adversary*. By this, we mean that the network is controlled by an adversary that, with full knowledge of the contents of messages, may choose to arbitrarily delay messages as long as between any two correct processes, messages are eventually delivered.

The goal of a Byzantine consensus protocol is to allow all correct processors to eventually *decide* some value. Specifically, a protocol that solves Byzantine consensus must satisfy:

Definition 1. *Agreement. If two correct processors decide, then they decide the same value. Also, if a correct processor decides more than once, it decides the same value each time.*

Definition 2. *Unanimity. If all correct processors have the same initial value v, then a correct processor that decides must decide v.*

Definition 3. *Validity. If a correct processor decides v, then v was the initial value of some processor.*

Definition 4. *Termination. All correct processors must eventually decide.*

Note that algorithms that satisfy all of the above requirements are not possible in asynchronous environments when even a single crash failure must be tolerated [10]. In practice, algorithms circumvent this limitation by assuming some limitation in the extent of asynchrony in the system, or by relaxing the Termination property to a probabilistic one where all correct processors terminate with probability 1.

Unanimity requires that the outcome be predetermined when the initial values of all correct processors are unanimous. A one-step algorithm takes advantage of such favorable initial conditions to allow correct processors to decide in one communication step.

We define two notions of one-step protocols:

Definition 5. *Strongly one-step. If all correct processors have the same initial value v, a* strongly one-step *Byzantine consensus algorithm allows all correct processors to decide v in one communication step.*

Definition 6. *Weakly one-step. If there are no faulty processors in the system and all processors have the same initial value v, a* weakly one-step *Byzantine consensus algorithm allows all correct processors to decide v in one communication step.*

While both can decide in one step, strongly one-step algorithms make fewer assumptions about the required conditions and in particular cannot be slowed down by Byzantine failures when all correct processors have the same initial value. Strongly one-step algorithms optimize for the case where some processors may be faulty, but there is no contention among correct processors, and weakly one-step algorithms optimize for cases that are both contention-free *and* failure-free.

3 Lower Bounds

We show that a Byzantine consensus algorithm that tolerates t Byzantine failures among n processors requires $n > 7t$ to be strongly one-step and $n > 5t$ to be

weakly one-step.[1] These results are for the best case scenario in which each correct processor broadcasts its initial value to all other processors in the first communication step and thus they hold for any algorithm.

3.1 Lower Bound for Strongly One-Step Byzantine Consensus

Lemma 1. *A strongly one-step Byzantine consensus algorithm must allow a correct processor to decide v after receiving the same initial value v from $n - 2t$ processors.*

Proof. Assume otherwise, that there exists a run in which a strongly one-step Byzantine algorithm $\mathcal{A}$ does not allow a correct processor p to decide v after receiving the same initial value v from $n - 2t$ processors. Since $\mathcal{A}$ is a strongly one-step algorithm, the fact that processor p does not decide after the first round implies that some correct processor q has an initial value v', $v' \neq v$. Now consider a second run, in which all correct processors *do* have the same initial value v. Without blocking, p can wait for messages from at most $n - t$ processors. Among these, t can be Byzantine and send arbitrary initial values. This means that processor p is only guaranteed to receive $n - 2t$ messages indicating that $n - 2t$ processors have the initial value v. Given that $\mathcal{A}$ is a strongly one-step algorithm, p must decide v at this point. However, from the point of view of p, this second run is indistinguishable from the first run. This is a contradiction. □

Theorem 1. *Any strongly one-step Byzantine consensus protocol that tolerates t failures requires at least $7t + 1$ processors.*

Proof. Assume that there exists a strongly one-step Byzantine consensus algorithm $\mathcal{A}$ that tolerates up to t Byzantine faults and requires only $7t$ processors. We divide the processors into three groups: G_0 and G_1 each contain $3t$ processors, of which the correct processors have initial values 0 and 1 respectively; G_* contain the remaining t processors.

Now consider the following configurations $\mathcal{C}_0$ and $\mathcal{C}_1$. In $\mathcal{C}_0$, t of the processors in G_1 are Byzantine, and processors in G_* have the initial value 0. Assume that Byzantine processors act as if they are correct processors with initial value 0 when communicating with processors in G_*, and initial value 1 when communicating with processors not in G_*. Now consider that a correct processor $p_0 \in G_*$ collects messages from $n - t$ processors in the first communication step. Given that the network adversary controls the order of message delivery, p_0 can be made to receive messages from all processors in G_0 and G_*, and the t Byzantine processors in G_1. p_0 thus receives $n - 2t$ messages indicating that the $n - 2t$ senders have initial value 0. By Lemma 1, p_0 must decide 0 after that first communication step. In order to satisfy Agreement, $\mathcal{A}$ must ensure that any correct processor that ever decides in $\mathcal{C}_0$ decides 0. We say that $\mathcal{C}_0$ is 0-*valent*.

In $\mathcal{C}_1$, t of the processors in G_0 are Byzantine, and processors in G_* have the initial value 1. In addition, Byzantine processors act as if they are correct

[1] These results are for threshold quorum systems, but may be generalized to use arbitrary quorum systems.

processors with initial value 1 when communicating with processors from G_* and initial value 0 when communicating with processors not in G_*. A correct processor $p_1 \in G_*$ collects messages from $n-t$ in the first communication step. Suppose that the network adversary chooses to deliver messages from G_1 and G_*, as well as from the t Byzantine processors. Now p_1 collects $n-2t$ messages indicating that $n-2t$ senders have initial value 1. By Lemma 1, p_1 must decide 1 after the first communication step. In order to satisfy Agreement, $\mathcal{A}$ must ensure that any correct processor that ever decides in $\mathcal{C}_1$ decides 1. We say that $\mathcal{C}_1$ is 1-*valent*.

Further assume that for both configurations, messages from any processor in G_* to any processor not in G_* are arbitrarily delayed such that in any asynchronous round, when a processor that is not in G_* awaits $n-t$ messages, it receives messages from every processor that is not in G_*. Now, any correct process $q_0 \notin G_*$ executing $\mathcal{A}$ in $\mathcal{C}_0$ will be communicating with $3t$ processors that behave as if they are correct processors with initial value 0 and $3t$ processors that behave as if they are correct processors with initial value 1. As we have shown above, $\mathcal{C}_0$ is a 0-valent configuration, so $\mathcal{A}$ must ensure that q_0 decides 0, if it ever decides. Similarly, a correct processor $q_1 \notin G_*$ executing $\mathcal{A}$ in $\mathcal{C}_1$ will also be communicating with $3t$ processors that behave as if they are correct processors with initial value 0 and $3t$ processors that behave as if they are correct processors with initial value 1. However, since we have shown that $\mathcal{C}_1$ is a 1-valent configuration, $\mathcal{A}$ must ensure that q_1 decides 1, even though it sees exactly the same inputs as q_0. This is a contradiction. □

3.2 Lower Bound for Weakly One-Step Byzantine Consensus

We now show the corresponding lower bound for weakly one-step algorithms. The lower bound for weakly one-step algorithms happens to be identical to that for two-step algorithms. The bound for two-step algorithms was shown in [11]. We show a corresponding bound for weakly one-step algorithm for completeness, but note that this is not a new result.

We weaken the requirement on Lemma 1 as follows:

Lemma 2. *A weakly one-step Byzantine consensus algorithm must allow a processor to decide v after learning that $n-t$ processors have the same initial value v.*

Proof. A processor can only wait for messages from $n-t$ processors without risking having to wait indefinitely. Since a weakly one-step Byzantine consensus algorithm must decide in one communication step if all correct processors have the same initial value and there are no Byzantine processors, it must decide if all of the $n-t$ messages claim the same initial value. □

Theorem 2. *A weakly one-step Byzantine consensus protocol that tolerates t failures requires at least $5t+1$ processors.*

Proof. We provide only a sketch of the proof since it is similar to that of Theorem 1. Proof by contradiction. Assume that a Byzantine consensus algorithm

$\mathcal{A}$ is weakly one-step and requires only $5t$ processors. We divide the $5t$ processors into three groups, G_0, G_1, and G_*, containing $2t$, $2t$, and t processors respectively. All correct processors in G_0 have the initial value 0 and all correct processors in G_1 have the initial value 1.

As in the proof of Theorem 1, we construct two configurations $\mathcal{C}_0$ and $\mathcal{C}_1$. In $\mathcal{C}_0$, processors in G_* have the initial value 0 and t processors in G_1 are Byzantine. Correspondingly, in $\mathcal{C}_1$, processors in G_* have the initial value 1 and t processors in G_0 are Byzantine. These Byzantine processors behave as they do in the proof of Theorem 1. It is thus possible for processors in G_* to decide 0 and 1 in $\mathcal{C}_0$ and $\mathcal{C}_1$ respectively. Therefore, correct processors in G_0 and G_1 must not decide any value other than 0 and 1 respectively. However, if all messages from any processor in G_* to any processor not in G_* are delayed, then correct processors in $\mathcal{C}_0$ and $\mathcal{C}_1$ see exactly the same inputs. This is a contradiction. □

4 Bosco

We now present Bosco (**B**yzantine **O**ne-**S**tep **CO**nsensus), an algorithm that meets the bounds presented in the previous section. To the best of our knowledge, Bosco is the first strongly one-step algorithm that solves asynchronous Byzantine consensus with optimal resilience. The idea behind Bosco is simple, and resembles the one presented in [2]. We simply extend the results of [2] to handle Byzantine failures. The Bosco algorithm is shown in Algorithm 1.

Algorithm 1. Bosco: a one-step asynchronous Byzantine consensus algorithm

Input: v_p
1 broadcast $\langle \texttt{VOTE}, v_p \rangle$ to all processors
2 wait until $n - t$ `VOTE` messages have been received
3 **if** *more than* $\frac{n+3t}{2}$ `VOTE` *messages contain the same value* v **then**
4 DECIDE(v)
5 **if** *more than* $\frac{n-t}{2}$ `VOTE` *messages contain the same value* v,
6 *and there is only one such value* v **then**
7 $v_p \leftarrow v$
8 `Underlying-Consensus`(v_p)

Bosco is an asynchronous Byzantine consensus algorithm that satisfies Agreement, Unanimity, Validity, and Termination. Bosco requires $n > 3t$, where n is the number of processors in the system, and t is the maximum number of Byzantine failures that can be tolerated, in order to provide these correctness properties. In addition, Bosco is weakly one-step when $n > 5t$ and strongly one-step when $n > 7t$.

The main idea behind Bosco is that if all processors have the same initial value, then given enough processors in the system, a correct processor is able to observe sufficient information to safely decide in the first communication round. Additional mechanisms ensure that if such an early decision ever happens, all

correct processors *must* either i) early decide the same value; or ii) set their local estimates to the value that has been decided.

When the algorithm starts, each processor p receives an input value v_p, that is the value that the processor is trying to get decided and the value that it will use for its local estimate. Each processor broadcasts this initial value in a `VOTE` message, and then waits for `VOTE` messages from $n - t$ processors (likely including itself). Since at most t processors can fail, votes from $n - t$ processors will eventually be delivered to each correct processor.

Among the votes that are collected, each processor checks two thresholds: if more than $\frac{n+3t}{2}$ of the votes are for some value v, then a processor decides v; if more than $\frac{n-t}{2}$ of the votes are for some value v, then a processor sets its local estimate to v. Each processor then invokes `Underlying-Consensus`, a protocol that solves asynchronous Byzantine consensus (satisfies Agreement, Unanimity, Validity, and Termination), but is not necessarily one-step.

We first prove that Bosco satisfies Agreement, Unanimity, Validity, and Termination, when $n > 3t$.

Lemma 3. *If two correct processors p and q decide values v and v' in line 4, then $v = v'$.*

Proof. Assume otherwise, that two correct processors p and q decide values v and v' in line 4 such that $v \neq v'$. p and q must have collected more than $\frac{n+3t}{2}$ votes for v and v' each. Since there are only n processors in the system, these two sets of votes share more than $\frac{3t}{2}$ common senders. Given that only t of these senders can be Byzantine, more of $\frac{t}{2}$ of these senders are correct processors. Since a correct processor must send the same vote to all processors (in line 1), $v = v'$. This is a contradiction. □

Lemma 4. *If a correct processor p decides a value v in line 4, then any correct processor q must set its local estimate to v in line 6.*

Proof. Assume otherwise, that a correct processor p decides a value v in line 4, and a correct processor q does not set its local estimate to v in line 6. Since processor p decides in line 4, it must have collected more than $\frac{n+3t}{2}$ votes for v in line 2. Since processor q does not set its local estimate to v in line 6, it must have collected no more than $\frac{n-t}{2}$ votes for v, or collected more than $\frac{n-t}{2}$ votes for some value v', $v' \neq v$. For the first case, consider that since there are only n processors in the system, processor q must have collected votes from at least $n - 2t$ of the senders that processor p collected from. Among these, more than $\frac{n+t}{2}$ sent a vote for v to q. Since at most t of these processors can be Byzantine, processor q must have received more than $\frac{n-t}{2}$ votes for v. This is a contradiction. For the second case, if q collects more than $\frac{n-t}{2}$ votes for some value v', $v' \neq v$, then more than t of these senders must be among those that sent a vote for v to processor q. This is a contradiction, since, no more than t of the processors in the system can be Byzantine. □

Theorem 3. *Bosco satisfies Agreement.*

Proof. There are two cases to consider. In the first case, no processor collects sufficient votes containing the same value to decide in line 4. This means that all decisions occur in `Underlying-Consensus`. Since `Underlying-Consensus` satisfies Agreement, Bosco satisfies Agreement. In the second case, some correct processor p decides some value v in line 4. By Lemma 3, any other processor that decides in line 4 must decide the same value. By Lemma 4, all correct processors must change their local estimates to v in line 6. Therefore, all correct processors will invoke `Underlying-Consensus` with the value v. Since `Underlying-Consensus` satisfies Unanimity, all correct processors that decide in `Underlying-Consensus` must also decide v. □

Theorem 4. *Bosco satisfies Unanimity.*

Proof. Proof by contradiction. Suppose a processor p decides v', but all correct processors have the same initial value v, $v' \neq v$. Since only t Byzantine processors can broadcast vote messages that contain $v \neq v'$, no correct processor can collect sufficient votes to either decide in line 4 or to set its local estimate in line 6. Therefore, in order for a processor to decide v, `Underlying-Consensus` must allow correct processors to decide v even though all correct processors start `Underlying-Consensus` with the initial value v'. This is a contradiction since `Underlying-Consensus` satisfies Unanimity. □

Theorem 5. *Bosco satisfies Validity.*

Proof. If a processor decides v in line 4, more than $\frac{n+3t}{2}$ processors voted v and more than $\frac{n+t}{2}$ of these processors are correct and had initial value v. Similarly, if a processor sets its local estimate to v in line 6, more than $\frac{n-t}{2}$ processors voted v and more than $\frac{n-3t}{2}$ of these processors are correct and had initial value v. Combined with the fact that `Underlying-Consensus` satisfies Validity, Bosco satisfies Validity. □

Note that satisfying Validity in general in a consensus protocol is non-trivial, particularly if the range of initial values is large. A thorough examination of the hardness of satisfying Validity is beyond the scope of this paper; we simply assume that `Underlying-Consensus` satisfies Validity for the range of initial values that it allows.

Theorem 6. *Bosco satisfies Termination.*

Proof. Since each processor awaits messages from $n - t$ processors in line 2, and there can only be t failures, line 2 is guaranteed not to block forever. Each processor will therefore invoke the underlying consensus protocol at some point. Therefore, Bosco inherits the Termination property of `Underlying-Consensus`. □

Next, we show that Bosco offers strongly and weakly one-step properties when $n > 7t$ and $n > 5t$ respectively.

Theorem 7. *Bosco is Strongly One-Step if* $n > 7t$.

Proof. Assume that all correct processors have the same initial value v. Now consider any correct processor that collects $n - t$ votes in line 2. At most t of these votes can be from Byzantine processors and contain values other than v. Therefore, all correct processors must obtain at least $n - 2t$ votes for v. Since $n > 7t$, $2n - 4t > n + 3t$. This means that $n - 2t > \frac{n+3t}{2}$. Therefore, all correct processors will collect sufficient votes and decide in line 4. □

Theorem 8. *Bosco is Weakly One-Step if* $n > 5t$.

Proof. Assume that there are no failures in the system and that all processors have the same initial value v. Then any correct processor must collect $n-t$ votes that contain v in line 2. Given that $n > 5t$, $2n - 2t > n + 3t$. This means that $n - t > \frac{n+3t}{2}$. Therefore, all correct processors will collect sufficient votes and decide in line 4. □

5 Discussion

One important feature of Bosco, from which it draws its simplicity, is its dependence on an underlying consensus protocol that it invokes as a subroutine. This allows the specification of Bosco to be free of complicated mechanisms typically found in consensus protocols to ensure correctness. While it is clear that any Byzantine fault-tolerant consensus protocol that provides Agreement, Unanimity, Validity, and Termination can be used for the subroutine in Bosco, the FLP impossibility result [10] states that such a protocol cannot actually exist! Two common approaches have been used to sidestep the FLP result: assuming partial synchrony or relaxing the termination property to a probabilistic termination property. Thankfully, such algorithms can be used as subroutines to Bosco, resulting in one-step algorithms that either require partial synchrony assumptions, or provide probabilistic termination properties (or both). An example of an algorithm that can be used as a subroutine in Bosco is the Ben-Or algorithm [12]. Algorithms that do not provide validity, such as PBFT [13], cannot be used by Bosco.

While abstracting away the underlying consensus protocol simplifies the specification and correctness proof of Bosco, for practical purposes it may be advantageous to unroll the subroutine. This potentially allows piggybacking of messages and improves the efficiency of implementations. As an example, Algorithm 2 shows RS-Bosco, a randomized strongly one-step version of Bosco which does not depend on any underlying consensus protocol. RS-Bosco is strongly one-step and requires that $n > 7t$. It does not satisfy Termination as defined in section 2, but instead provides Probabilistic Termination:

Definition 7. *Probabilistic Termination. All correct processors decide with probability 1.*

Algorithm 2. RS-Bosco: a randomized strongly one-step asynchronous Byzantine consensus algorithm

```
1  Initialization
2    x_p ← v_p
3    r_p ← 0
4  Round r_p
5    Broadcast ⟨VOTE, r_p, x_p⟩ to all processors
6    Collect n − t ⟨VOTE, r_p, ∗⟩ messages
7    if more than (n+3t)/2 VOTE msgs contain v then
8        DECIDE(v)
9    if more than (n−t)/2 VOTE msgs contain v then
10       Broadcast ⟨CANDIDATE, r_p, v⟩
11   else
12       Broadcast ⟨CANDIDATE, r_p, ⊥⟩
13   end
14   Collect n − t ⟨CANDIDATE, r_p, ∗⟩ messages
15   if at least t + 1 msgs are NOT of the form ⟨CANDIDATE, r_p, x_p⟩ then
16       x_p ←RANDOM() // pick randomly from {0,1}
17   r_p ← r_p + 1
```

For brevity, the proof of correctness of RS-Bosco is omitted. We note that RS-Bosco suffers from two limitations as currently constructed. First, RS-Bosco solves only binary consensus. Second, RS-Bosco uses a local coin to randomly update local estimates when a threshold of identical votes cannot be obtained. This mechanism is similar to that in the Ben-Or algorithm and causes the algorithm to require an exponential number of rounds for decision when contention is present. We believe that these limitations can be overcome in practical implementations, but a thorough discussion is beyond the scope of this paper.

6 Related Work

One-step consensus algorithms for crash failures have previously been studied. Brasileiro et al. [2] proposed a general technique for converting any crash-tolerant consensus algorithm into a crash-tolerant consensus algorithm that terminates in one communication step if all correct processors have the same initial value. Bosco is an extension of the ideas presented in that work to handle Byzantine failures. The key difference between handling crashed failures and Byzantine failures is that when Byzantine failures need to be tolerated, equivocation must be handled correctly.

A simple and elegant crash-tolerant consensus algorithm of the same flavor, One-Third-Rule, appears in [4]. This work has been extended to handle Byzantine faults by considering transmission faults where messages can be corrupted in addition to being dropped [14]. The algorithms in [4,14] differ from the

algorithms we have presented because they use a different failure model, where failures are attributed to message transmissions, rather than to processors.

Friedman et al. [8] proposed a weakly one-step algorithm that tolerates Byzantine faults and terminates with probability 1 but does not tolerate a strong network adversary. In particular, their protocol is dependent on a common coin oracle and assumes that the network adversary has no access to this common coin; a strong network adversary with access to the common coin can prevent termination. In comparison, Bosco does not explicitly depend on any oracles, although the subroutine invoked by Bosco may have such dependencies. With a judicious choice of the consensus subroutine, Bosco can tolerate a strong network adversary that can arbitrarily re-order messages and collude with Byzantine processors. In particular, RS-Bosco does not require any oracles and tolerates strong network adversaries.

Zielinski [15] presents a framework for expressing various consensus protocols using an abstraction called *Optimistically Terminating Consensus* (OTC). Among the algorithms constructed by Zielinski are two Byzantine consensus algorithms with one-step characteristics that require $n > 5t$ and $n > 3t$. The first of these algorithms is a weakly one-step algorithm that requires partial synchrony; the second algorithm, while appearing to violate the lower bounds we have shown in this paper, is neither weakly nor strongly one-step because processors can only decide in the first communication step when, in addition to the system being failure-free and contention-free, all processors are fast enough that the timeout mechanism in the algorithm is not triggered.

Many techniques have been proposed to improve the performance and reduce the overhead of providing Byzantine fault tolerance. Abd-El-Malek et al. [16] proposed the optimistic use of quorums rather than agreement protocols to obtain higher throughput. However, in the face of contention, optimistic quorum systems perform poorly. HQ combines the use of quorums and consensus techniques to provide high performance during normal operation and minimize overhead during periods of contention [17]. Probabilistic techniques have also been proposed to reduce the overhead of using quorum systems to provide Byzantine fault-tolerance [18,19]. Hendricks et al. [20] proposed the use of erasure coding to minimize the overhead of a Byzantine fault-tolerant storage system. Zyzzyva, another recently proposed Byzantine fault-tolerant system, uses optimistic speculation to decrease the latency observed by clients [21]. In comparison, the one-step Byzantine consensus algorithms presented in this paper aims to improve performance by exploiting contention-free and failure-free situations to provide decisions in one communication step.

Lamport [5] presents lower bounds for the number of message delays and the number of processors needed for several kinds of asynchronous non-Byzantine consensus algorithm in; in particular, *Fast Learning* algorithms are one-step algorithms for non-Byzantine settings. A one-step version of Paxos [22], Fast Paxos, is presented in [3,6]. Fast Paxos tolerates only crash failures, although [6] alludes to the possibility of a Byzantine fault-tolerant version of Fast Paxos.

7 Conclusion

Byzantine fault tolerance has drawn significant interest from both academia and the industry recently. While Byzantine fault tolerance aims to provide resilience against arbitrary failures, in many applications, failures and contention are not the norm. This paper explores optimization opportunities in contention-free and failure-free situations.

Overall, this paper makes three contributions: 1) we provide two definitions of one-step asynchronous Byzantine consensus algorithms that provide low latency performance in favorable conditions while guaranteeing strong consistency when failures and contention occur; 2) we prove lower bounds in the number of processors required for such algorithms; and 3) we present Bosco, a one-step algorithm for Byzantine asynchronous consensus that meets these bounds.

References

1. Keidar, I., Rajsbaum, S.: On the cost of fault-tolerant consensus when there are no faults. SIGACT News 32(2), 45–63 (2001)
2. Brasileiro, F.V., Greve, F., Mostéfaoui, A., Raynal, M.: Consensus in one communication step. In: Proc. of the 6th International Conference on Parallel Computing Technologies, pp. 42–50. Springer, London (2001)
3. Boichat, R., Dutta, P., Frolund, S., Guerraoui, R.: Reconstructing Paxos. ACM SIGACT News 34 (2003)
4. Charron-Bost, B., Schiper, A.: The Heard-Of model: Unifying all benign failures. Technical Report LSR-REPORT-2006-004, EPFL (2006)
5. Lamport, L.: Lower bounds for asynchronous consensus. Technical Report MSR-TR-2004-72, Microsoft Research (2004)
6. Lamport, L.: Fast Paxos. Distributed Computing 19(2), 79–103 (2006)
7. Dobre, D., Suri, N.: One-step consensus with zero-degradation. In: DSN 2006: Proceedings of the International Conference on Dependable Systems and Networks, pp. 137–146. IEEE Computer Society, Washington (2006)
8. Friedman, R., Mostefaoui, A., Raynal, M.: Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems. IEEE Transactions on Dependable and Secure Computing 2(1), 46–56 (2005)
9. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Transactions on Programming Languages and Systems 4(3), 382–401 (1982)
10. Fischer, M., Lynch, N., Patterson, M.: Impossibility of distributed consensus with one faulty process. J. ACM 32(2), 374–382 (1985)
11. Martin, J.P., Alvisi, L.: Fast Byzantine consensus. In: Proceedings of the International Conference on Dependable Systems and Networks, pp. 402–411 (June 2005)
12. Ben-Or, M.: Another advantage of free choice: Completely asynchronous agreement protocols. In: Proc. of the 2nd ACM Symp. on Principles of Distributed Computing, Montreal, Quebec, ACM SIGOPS-SIGACT, pp. 27–30 (August 1983)
13. Castro, M., Liskov, B.: Practical Byzantine fault tolerance. In: Proc. of the 3rd Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, LA (February 1999)

14. Biely, M., Widder, J., Charron-Bost, B., Gaillard, A., Hutle, M., Schiper, A.: Tolerating corrupted communication. In: PODC 2007: Proceedings of the twenty-sixth annual ACM symposium on Principles of Distributed Computing, pp. 244–253. ACM, New York (2007)
15. Zielinski, P.: Optimistically terminating consensus: All asynchronous consensus protocols in one framework. In: ISPDC '06: Proceedings of the Proceedings of The Fifth International Symposium on Parallel and Distributed Computing, Washington, DC, pp. 24–33. IEEE Computer Society Press, Los Alamitos (2006)
16. Abd-El-Malek, M., Ganger, G.R., Goodson, G.R., Reiter, M.K., Wylie, J.J.: Fault-scalable Byzantine fault-tolerant services. SIGOPS Operating Systems Review 39(5), 59–74 (2005)
17. Cowling, J., Myers, D., Liskov, B., Rodrigues, R., Shrira, L.: HQ replication: a hybrid quorum protocol for Byzantine fault tolerance. In: OSDI 2006: Proceedings of the 7th symposium on Operating Systems Design and Implementation, pp. 177–190. USENIX Association, Berkeley (2006)
18. Merideth, M.G., Reiter, M.K.: Probabilistic opaque quorum systems. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 403–419. Springer, Heidelberg (2007)
19. Malkhi, D., Reiter, M.K., Wool, A., Wright, R.N.: Probabilistic quorum systems. Information and Computation 170(2), 184–206 (2001)
20. Hendricks, J., Ganger, G.R., Reiter, M.K.: Low-overhead Byzantine fault-tolerant storage. In: Proc. of twenty-first ACM SIGOPS Symposium on Operating Systems Principles, pp. 73–86. ACM, New York (2007)
21. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.: Zyzzyva: speculative Byzantine fault tolerance. In: Proc. of twenty-first ACM SIGOPS symposium on Operating Systems Principles, pp. 45–58. ACM, New York (2007)
22. Lamport, L.: The part-time parliament. Trans. on Computer Systems 16(2), 133–169 (1998)