Doubly-Expedited One-Step Byzantine Consensus

Nazreen Banu, Taisuke Izumi, Koichi Wada

Graduate school of Engineering,
Nagoya Institute of Technology, Gokisho-cho,
Showa-ku, Nagoya, Aichi, 466-8555, Japan.
Phone: +81-52-735-5408, Fax: +81-52-735-5408
nazreentech@phaser.elcom.nitech.ac.jp,{t-izumi,wada}@nitech.ac.jp

April 9, 2010

Abstract

It is known that Byzantine consensus algorithms guarantee one-step decision only in favorable situations (e.g. when all processes propose the same value) and no one-step algorithm can support two-step decision. This paper presents *DEX*, a novel one-step Byzantine algorithm that circumvents these impossibilities using the condition-based approach. Algorithm *DEX* has two distinguished features: Adaptiveness and Double-expedition property. Adaptiveness makes it sensitive to only actual number of failures so that it provides fast termination for more number of inputs when there are fewer failures (a common case in practice). The double-expedition property facilitates two-step decision in addition to one-step decision by running two condition-based mechanisms in parallel. To the best of our knowledge, double-expedition property is the new concept introduced by this paper, and *DEX* is the first algorithm having such a feature. Although *DEX* takes four steps at worst in well-behaved runs while existing one-step algorithms take only three, it is expected to work efficiently because the worst-case does not occur so often in practice.

1 Introduction

1.1 Background

The *consensus* problem is central for the construction of fault-tolerant distributed systems. In the consensus problem, each process proposes a value, and all non-faulty processes have to agree on a common value which is one of the proposed values. Several practical agreement problems (such as atomic broadcast, view synchrony, state-machine replication, etc.) can be implemented using a solution to the consensus problem, and hence solving consensus is crucial in designing distributed systems.

The consensus problem has been studied with various failure models and different synchrony assumptions. This paper studies the consensus in asynchronous systems subject to Byzantine failures, i.e., faulty processes can behave in an arbitrary way, and there are assumptions neither about relative speed of processes nor about timely delivery of messages.

To converge on a single decision value, consensus protocols need to exchange messages. Each message exchange constitutes a communication step. The number of communication steps taken for reaching agreement is an important measure to evaluate the efficiency of the consensus algorithms. A previous work has proved that any consensus algorithm requires at least two communication steps for decision even in failure-free executions [9]. This lower bound often becomes a dominant part of the performance overhead imposed to consensus-based applications. However, this fact does not imply that the two-step lower bound is incurred for every input (an input to consensus algorithms is defined as a n-tuple consisting of all proposed values). For example, it does not hold for the case where all processes propose the same value. Furthermore, in typical runs of consensus-based applications, the consensus algorithm often receives such "good" inputs. For instance, consider a replicated state machine: The replicated servers need to agree on the processing order of the update requests. If a client broadcasts its request to all servers and there is no contention, then all servers propose the same request as the candidate they will handle next. Practically, it is not so often that two or more concurrent update-requests arise for the same data object.

This observation induces an interest in one-step decision for good inputs. The attempts to circumvent two-step lower bound are initiated by Brasileiro et al.[2]. They propose a general framework to convert any crash-tolerant algorithm into the one that solves the consensus for any input, and especially terminates in one step when all processes propose the same value. In the following results [3, 4], the notion of one-step decision is considered in combination with other schemes such as randomization and failure detectors.

An interesting aspect of one-step decision schemes is to characterize the situations where one-step decision is possible. The first investigation in that aspect is considered by Mostefaoui et.al.[11], which applies the *condition-based approach* for obtaining a good one-step decision scheme. In general, the condition-based approach defines a set of inputs, called *condition*, for which the condition-based algorithm guarantees a certain kind of good property. The first result with this approach[11] gives a sufficient class of conditions such that we can construct the condition-based algorithm guaranteeing one-step decision for any input belongs to the condition. This result is extended by Izumi and Masuzawa[8]. It gives the complete characterization of conditions that makes one-step decision possible.

While all of the above results are considered on crash-failure model, a recent work (BOSCO)[12] has devised one-step consensus algorithms on Byzantine failure model. It has shown two variants of one-step Byzantine consensus problem, weak and strong ones. The weakly one-step guarantees one-step decision only when all processes propose the same value and no process is faulty, but the strongly one-step must guarantee it in any situation only with a common proposed value, regardless of the number of faulty processes. In addition, this result also proposed an algorithm for each variant and proved that the assumptions $n > 5t$ and $n > 7t$ are necessary for weakly and strongly one-step Byzantine consensus respectively, where n is the number of processes and t is the maximum number of faulty processes.

1.2 Our Contribution

As seen above, the research challenge centered in one-step consensus is to enhance and clarify the situations favoring one-step decision. In the same research direction, this paper also explores Byzantine consensus problems with better one-step decision schemes. In particular, we focus on two features for one-step decision schemes shown as follows:

Adaptive Condition-Based Approach Most of the one-step algorithms are designed with the aim that they never violate agreement even if the number of failures is at the maximum. However, such a design works as a pessimistic approach when actual number of faulty processes is small, which is the usual case in real systems. An interesting way to circumvent this drawback is the use of *adaptive condition-based approach*. Informally, the adaptive condition-based approach handles the condition that dynamically changes according to the actual number of faulty processes (typically, fewer faults allow the condition with more number of inputs). In the context of one-step consensus, it means that the algorithm can terminate in one-step for more number of inputs when fewer processes are faulty. The notion of the adaptive condition-based approach is first introduced by Izumi and Masuzawa[7] and applied to one-step consensus problem in crash-failure model[8]. However, there is no result to apply it in Byzantine-failure model.

Double Expedition of One-Step Consensus One of the serious drawbacks in one-step decision schemes is the impossibility of zero-degradation[6, 3]. Informally, the zero-degradation is an important feature of consensus algorithms based on failure detectors, which always guarantees the best complexity (i.e., two steps) in *stable* runs where the failure detector does not make any mistakes and its output is stable. This impossibility result says that, to achieve one-step decision, any algorithm must sacrifice the

Table 1: Performance comparison of DEX with the existing works.

	System Model	Failure Type	Number of Processes	Feasibility for One-step Decision	Feasibility for Two-step Decision
Brasileiro et.al.[2]	Asyn.	Crash	3t+1	Agreed proposals	–
Izumi et.al.[8]	Asyn.	Crash	3t+1	Condition-Based	–
Mostefaoui et.al.[11]	Syn.	Crash	t+1	Condition-Based	–
Friedman et.al.[5]	Asyn.	Byzan.	7t+ 1	Agreed proposals	–
Yee et.al. [12] (Bosco)	Asyn.	Byzan.	5t+1 (Weak)	Agreed proposals	–
			7t+ 1 (Strong)	Agreed proposals of correct processes	–
DEX	Asyn.	Byzan.	6t+1	Condition-Based	Condition-Based

decision at the end of the second step. It is also shown that achieving both one-step decision and zero-degradation needs more stronger assumption about failure detections such as the existence of eventually perfect failure detector $\diamond P$. However, similar to the impossibility result of one-step decision [9], this result does not necessarily imply the impossibility of two-step decision for any inputs. Thus, it yields an interest to realize a *doubly-expedited* consensus algorithm, which equips a "conditional" two-step decision scheme combined with one-step decision.

This paper contributes a doubly-expedited Byzantine consensus algorithm based on the adaptive condition-based approach. The distinguished features of the proposed algorithm can be summarized as follows:

- In our construction, we first show a generic framework of the algorithm based on the notion of adaptive condition-based approach. Generally, in condition-based approach, adaptiveness property can be characterized by a *condition sequence*, which is defined as a sequence of $t + 1$ conditions such that the k-th ($0 \leq k \leq t$) condition is valid when the actual number of faulty processes is k. To attain the double-expedition property in adaptive manner, the framework is instantiated with a pair of condition sequences, each of which corresponds to the situations of one-step and two-step decisions respectively. We also show sufficient criteria, say *legality*, for the condition-sequence pair such that the doubly-expedited algorithms can be instantiated using any condition-sequence pair satisfying them.
- Two examples of legal condition-sequence pair, called *frequency-based pair* and *privileged-value-based pair*, are proposed. They have distinct advantages in the sense that the expedited situations corresponding to each pair is complementary. Interestingly, the algorithm instantiated by the frequency-based pair has more chances to decide in one or two steps compared to the existing one-step Byzantine consensus algorithms.Table 1 compares this result with the previous results.
- One drawback of the proposed framework is that it trades the decision scheme at third step for double-expedition property. This drawback causes a performance degradation in consensus-based applications when we consider pessimistic runs (that is, the given input is out of the conditions). However, standing on its optimistic counterpart, we make more inputs belong to the conditions so that our algorithm decides in one or two steps for many cases and achieves better performance in average.

To the best of our knowledge, double expedition property is the concept newly introduced in this paper. Hence, this paper is the first result showing the feasibility for taking both one- and two-step decision schemes simultaneously with no help of additional stronger assumptions.

1.3 Roadmap

The paper is organized as follows: Section 2 presents the system model, the definitions of Byzantine consensus problem and other necessary formalizations. Section 3 provides the legality criteria for doubly-expedited condition-sequence pair and two examples for it. Section 4 describes the generic framework of doubly-expedited one-step consensus algorithms. Section 5 provides the final remarks.

2 Preliminaries

2.1 System model

We assume an asynchronous distributed system model subject to Byzantine failures. It consists of a set $\prod = \{p_1, p_2, \ldots, p_n\}$ of n processes. Each process communicates with each other process by sending messages over a reliable link where neither message loss, duplication nor corruption occurs. Besides, there is no assumption about relative speed of processes or about the timely delivery of messages.

As we consider Byzantine failure model, a faulty process can behave arbitrarily, which means that it is allowed even not to follow the deployed algorithm. A process that is not faulty is said to be *correct*. We assume an upper bound, denoted by t, on the number of faulty processes. Each process knows the value of t in advance. Throughout this paper, we assume $5t < n$, which is the necessary assumption to make one-step decision possible. We also denote by f, the actual number of failures during executions. Notice that, no process can be aware of the value of f.

2.2 Byzantine Consensus and Underlying Consensus Primitive

The Byzantine consensus problem has been informally stated in the introduction: Each process proposes a value, and all correct processes have to decide a common value which is proposed by at least one process. Formally, the Byzantine consensus can be defined by the following requirements.

Termination Each correct process eventually decides a value.

Agreement If two correct processes decide, they must decide the same value.

Unanimity If all correct processes propose the same value v, then no correct process decides the value different from v.

In general, the consensus problem can not be solved in asynchronous system with no additional assumption. Hence, we need some assumptions to guarantee correct termination for arbitrary inputs. Many kinds of assumptions, such as partial synchrony, failure detectors, etc., are considered in past literatures. As our research direction is finding the feasibility of one-step decision, we simply assume an abstraction of them. More precisely, the system is assumed to be equipped with the *underlying consensus primitive* that ensures agreement, termination and unanimity, but provides no guarantees about its running time.

2.3 Condition-Based Approach

In the condition-based approach, an input vector is a n- dimensional vector, whose i-th entry contains the value proposed by a process p_i. Note that, since a faulty process can propose different values to distinct processes, the entries correspond to Byzantine processes are regarded to contain meaningless values. A condition defined for n processes is a subset of all possible input vectors. Adaptiveness in the condition-based approach is a property that allows a condition to change dynamically according to the actual number of faulty processes. That is, the fewer number of failures allow the condition with more number of input vectors. Thus, adaptiveness is defined by a *condition sequence* $(C_0, C_1, ...C_k...C_t)$ satisfying $C_k \supseteq C_{k+1}$ for any $k(0 \leq k \leq t-1)$, where the k-th condition corresponds to the set of input vectors that is valid when actual number of faults is equal to k.

2.4 Doubly-Expedited Consensus

In this subsection, we introduce a novel feature of consensus algorithms, called *double-expedition* property. In the execution of doubly-expedited algorithms, each process has two chances for fast decision by running both one-step and two-step decision schemes concurrently. Since both decision schemes guarantee fast decision only for good inputs, we characterize their property using a pair of condition sequences. We introduce the condition-sequence pair as follows: $(S^1, S^2) = ((C^1_0, C^1_1, \cdots C^1_k, \cdots C^1_t), (C^2_0, C^2_1, \cdots C^2_k, \cdots C^2_t))$, where S^1 and S^2 correspond to the condition sequences that identify the situations guaranteeing one-step and two-step decisions respectively. For example, consider the input vector I such that $I \notin C^1_k$, $I \in C^1_{k-1}$ and $I \in C^2_k$ hold. Then, if I is given to the consensus algorithm and the actual number of faulty processes is less than k, all processes decide in one step because $I \in C^1_{k-1}$. If (exactly) k processes are faulty, then one-step decision is no more guaranteed, but all processes necessarily decide in two steps because of $I \in C^2_k$.

3 Legality for Double Expedition

It is clear that we cannot design the doubly-expedited consensus algorithm for any pair of condition sequences. In this section, we propose sufficient criteria such that we can construct the doubly-expedited algorithm characterized by any condition-sequence pair satisfying them. In addition, we show two examples of condition-sequence pair that satisfy these criteria.

3.1 Notations

Let $\mathcal{V}$ be an ordered set of all possible proposal values. We introduce the default value $\perp$ not in $\mathcal{V}$. Letting I be an input vector in $\mathcal{V}^n$, we define a *view* J of I to be a vector in $(\mathcal{V} \cup \{\perp\})^n$ which is obtained by replacing at most t entries in I by $\perp$ and denote $J[k]$ as the k-th element of J. As well as, we define $\perp^n$ be a vector with all entries equal to $\perp$. The number of occurrences of value v in a view J is denoted by $\#_v(J)$. For two views J_1 and J_2, the containment relation $J_1 \leq J_2$ is defined as $\forall k (1 \leq k \leq n) : J_1[k] \neq \perp \Rightarrow J_1[k] = J_2[k]$. Also, for two views J_1 and J_2, let $dist(J_1, J_2)$ be the Hamming distance between J_1 and J_2 (that is, $dist(J_1, J_2) = |\{k \in \{1, 2, ..n\} | J_1[k] \neq J_2[k]\}|$). As well as $\mathcal{V}^n_k$ denotes the set of all views where $\perp$ values appears at most k times. The number of non-default values in J is denoted by $|J|$.

3.2 Legality Criteria

Given a pair of condition sequences $(S^1 S^2)$, we consider two predicates $P1, P2 : \mathcal{V}^n_t \rightarrow \{\text{True}, \text{False}\}$[1] and a function $F : \mathcal{V}^n_t \rightarrow \mathcal{V}$. Then, (S^1, S^2) is said to be *legal* if we can define $P1$, $P2$ and F satisfying the following five properties:

- LT1 : $\forall J \in \mathcal{V}^n_t : \exists I : I \in C^1_k \wedge dist(J, I) \leq k \Rightarrow P1(J)$.
- LT2 : $\forall J \in \mathcal{V}^n_t : \exists I : I \in C^2_k \wedge dist(J, I) \leq k \Rightarrow P2(J)$.
- LA3 : $\forall J, J' \in \mathcal{V}^n_t : P1(J) \wedge \exists I, I' : J \leq I \wedge J' \leq I' \wedge dist(I, I') \leq t \Rightarrow F(J) = F(J')$.
- LA4 : $\forall J, J' \in \mathcal{V}^n_t : P2(J) \wedge \exists I : J \leq I \wedge J' \leq I \Rightarrow F(J) = F(J')$.
- LU5 : $\forall J \in \mathcal{V}^n_t \Rightarrow F(J) = a : \#_a(J) > t \vee F(J) =$ the most common non $\perp$ value in J.

These properties are used to enforce the basic requirements of the doubly-expedited Byzantine consensus. Informally, $P1$ and $P2$ are the predicates to test whether the current view of a process contains

[1]In what follows $P1(J) = $ true is abbreviated as $P1(J)$

sufficient information to decide in one or two step(s) respectively, and F is the function to obtain the decision value from the current view. Thus, the first property *LT1* is for imposing one-step termination. The predicate *P1* must allow each correct process to decide in one step if its view has the possibility to come from an input vector included in the condition C^1_k and the actual number of failures is less than or equal to k. Similarly, the property *LT2* corresponds to two-step decision. The property *LA3* (or *LA4*) enforces the agreement between one-step (or two-step) decision and others. The last property *LU5* is the one to guarantee unanimity.

3.3 Example 1: Frequency-Based Legal Condition-sequence pair

This subsection introduces a legal condition-sequence pair (P^{freq}) that is based on the frequency-based condition and proves its legality. Let $1st(J)$ be a non $\perp$ value that appears most often in a vector J. If two or more values appear most often in J, then the largest one is selected. Let $\hat{J}$ be a vector obtained by replacing $1st(J)$ from J by $\perp$, and we define $2nd(J) = 1st(\hat{J})$. That is, $2nd(J)$ is the second most frequent value in J. The frequency-based condition C^{freq}_d is defined as follows:

$$C^{freq}_d = \{I \in \mathcal{V}^n | \#_{1st(I)}(I) - \#_{2nd(I)}(I) > d\}$$

It is known that C^{freq}_d belongs to *d-legal* conditions [10], which are necessary and sufficient to solve the consensus in failure prone asynchronous systems, where at most d processes can crash.

Using this condition, we can construct the frequency-based condition-sequence pair (P^{freq}) as follows:

$$P^{freq} = (S^1, S^2) = ((C^1_0, C^1_1, C^1_2, ...C^1_k, ...C^1_t),\ (C^2_0, C^2_1, C^2_2, ...C^2_k, ...C^2_t))$$

where

$$C^1_k = C^{freq}_{4t+2k} \text{ and } C^2_k = C^{freq}_{2t+2k}$$

As well as, the associated parameters $P1^{freq}$, $P2^{freq}$ and F^{freq} can be defined as below:

- $P1^{freq}(J) \equiv \#_{1st(J)}(J) - \#_{2nd(J)}(J) > 4t$.
- $P2^{freq}(J) \equiv \#_{1st(J)}(J) - \#_{2nd(J)}(J) > 2t$.
- $F^{freq}(J) = 1st(J)$.

Notice that, since there are at most t Byzantine processes, the stronger assumption $n > 6t$ is required to construct P^{freq}.

Theorem 1 The condition-sequence pair P^{freq} is legal.

Proof *LT1:* We have to show that $\forall J \in \mathcal{V}^n_t : \exists I \in C^1_k \wedge dist(J, I) \leq k \Rightarrow P1(J)$.

That is $\#_{1st(I)}(I) - \#_{2nd(I)}(I) > 4t + 2k \wedge dist(J, I) \leq k \Rightarrow \#_{1st(J)}(J) - \#_{2nd(J)}(J) > 4t$.

Since $I \in C^1_k$, we have $\#_{1st(I)}(I) - \#_{2nd(I)}(I) > 4t + 2k$. As $dist(J, I) \leq k$, we get $\#_{1st(I)}(J) \geq \#_{1st(I)}(I) - k$. Also, for any value $x \neq 1st(I)$, $\#_x(J) \leq \#_x(I) + k$ holds. Since $2nd(I)$ is the second most frequent value in I, $\#_x(J) \leq \#_{2nd(I)}(I) + k$. Hence, $\#_{1st(I)}(J) - \#_x(J) \geq \#_{1st(I)}(I) - k - \#_{2nd(I)}(I) - k$. Therefore, we get $\#_{1st(I)}(J) - \#_x(J) > 4t$. It implies that $1st(I) = 1st(J)$ and $\#_{1st(J)}(J) - \#_{2nd(J)}(J) > 4t$.

LT2: We have to show that $\forall J \in \mathcal{V}^n_t : \exists I \in C^2_k \wedge dist(J, I) \leq k \Rightarrow P2(J)$.

That is $\#_{1st(I)}(I) - \#_{2nd(I)}(I) > 2t + 2k \wedge dist(J, I) \leq k \Rightarrow \#_{1st(J)}(J) - \#_{2nd(J)}(J) > 2t$.

The proof is almost the same as the proof *LT1* (with only replacing C^1_k and $4t$ with C^2_k and $2t$ respectively).

LA3: Consider $J, J' \in \mathcal{V}_t^n$. We have to show that if $P1^{freq}(J) \wedge \exists I, I' : J \leq I \wedge J' \leq I' \wedge dist(I, I') \leq t$, then $1st(J) = 1st(J')$.

Suppose $1st(J) \neq 1st(J')$ for contradiction. Since $P1^{freq}(J)$ holds, $\#_{1st(J)}(J) - \#_v(J) > 4t$ also holds for any value $v \neq 1st(J)$. From the facts $J \in \mathcal{V}_t^n$ and $J \leq I$, it follows that at most t entries of I, which are occupied by the value $\perp$ in J, can contain v. Hence, $\#_{1st(J)}(I) - \#_v(I) > 3t$ holds. It implies that $1st(I) = 1st(J)$. Then, since $dist(I, I') \leq t$, I and I' can differ in at most t entries. Let these entries contain $1st(J)$ and v respectively in I and I'. Hence, $\#_{1st(J)}(I') \geq \#_{1st(J)}(I) - t$ and $\#_v(I') \leq \#_v(I) + t$. Therefore, $\#_{1st(J)}(I') - \#_v(I') \geq \#_{1st(J)}(I) - t - \#_v(I) - t$. When simplified, we get $\#_{1st(J)}(I') - \#_v(I') > t$. This implies that $1st(I') = 1st(I) = 1st(J)$. Then, from $J' \in \mathcal{V}_t^n$ and $J' \leq I'$, it follows that at most t entries of J', which corresponds to $1st(J)$ in I', can contain $\perp$. It implies that $\#_{1st(J)}(J') \geq \#_{1st(J)}(I') - t$ and $\#_v(J') \leq \#_v(I')$. It follows that $\#_{1st(J)}(J') - \#_v(J') > 0$, and thus we get $1st(J) = 1st(J')$.

LA4: Consider $J, J' \in \mathcal{V}_t^n$. It suffices to show that if $P2^{freq}(J) \wedge \exists I : J \leq I \wedge J' \leq I$, then $1st(J) = 1st(J')$.

Since $P2^{freq}(J)$ holds, $\#_{1st(J)}(J) - \#_v(J) > 2t$ also holds for any $v \neq 1st(J)$. As $J \leq I$ and $J \in \mathcal{V}_t^n$, at most t entries of I, which are occupied by $\perp$ in J, can contain the value v. Hence, $\#_v(I) \leq \#_v(J) + t$. This implies that $\#_{1st(J)}(I) - \#_v(I) > t$ holds. Therefore, $1st(J) = 1st(I)$. Similarly, since $J' \leq I$ and $J' \in \mathcal{V}_t^n$, at most t entries of J', which are occupied by the value $1st(J)$ in I, can contain $\perp$. It follows that $\#_{1st(J)}(J') \geq \#_{1st(J)}(I) - t$ and $\#_v(J') \leq \#_v(I)$. Thus, we get $\#_{1st(J)}(J') - \#_v(J') > 0$. This implies that $1st(J) = 1st(J')$.

LU5: This property is trivially satisfied since $1st(J)$ is the most frequent non $\perp$ value in J. □

3.4 Example 2: Privileged-Value-Based Condition-Sequence Pair

In this subsection, we present another legal condition-sequence pair (P^{prv}) constructed from a privileged-value-based condition, and prove its legality. In some practical agreement problems such as atomic commitment, a single value (e.g., Commit) is often proposed by most of the processes. The previous results [2, 6] have shown that, if this value is assigned some privilege, it is possible to expedite the decision. Let us assume that there is a value (say m) that is privileged among the set of all proposal values. Each process knows the value m a priori. Then, the privileged-value-based condition $C_d^{prv(m)}$ can be defined as follows:

$$C_d^{prv(m)} = \{I \in \mathcal{V}^n | \#_m(I) > d\}$$

Note that $C_d^{prv(m)}$ also belongs to *d-legal* conditions [10], which are necessary and sufficient to solve the consensus in failure prone asynchronous systems where at most d processes can crash.

Using this condition, we can construct the privileged-value-based condition-sequence pair P^{prv} as follows:

$$P^{prv} = (S^1, S^2) = ((C_0^1, C_1^1, C_2^1, ...C_k^1, ...C_t^1), (C_0^2, C_1^2, C_2^2, ...C_k^2, ...C_t^2))$$

where

$$C_k^1 = C_{3t+k}^{prv(m)} \text{ and } C_k^2 = C_{2t+k}^{prv(m)}$$

Also, we define the related parameters $P1^{prv}$, $P2^{prv}$ and F^{prv} as follows:

- $P1^{prv}(J) \equiv \#_m(J) > 3t$.
- $P2^{prv}(J) \equiv \#_m(J) > 2t$.

- $$F^{prv}(J) \equiv \begin{cases} m & \text{if } \#_m(J) > t \\ \begin{array}{l}\text{the most freq.}\\ \text{non default val.in } J\end{array} & \text{otherwise} \end{cases}$$

Notice that, since there are at most t Byzantine processes, the assumption $n > 5t$ is required to make P^{prv} meaningful.

Theorem 2 The condition-sequence pair P^{prv} is legal.

Proof *LT1:* We have to show that if $\#_m(I) > 3t + k \wedge dist(J, I) \leq k$, then $\#_m(J) > 3t$ holds.

Let I satisfies $\#_m(I) > 3t+k$. Since $dist(J, I) \leq k$, $\#_m(J) \geq \#_m(I)-k$. Hence, $\#_m(J) > 3t+k-k$. Thus we obtain $\#_m(J) > 3t$.

LT2: We have to show that if $\#_m(I) > 2t + k \wedge dist(J, I) \leq k$, then $\#_m(J) > 2t$ holds.

This proof is almost the same as the proof of *LT1* (with only replacing C_k^1 and $3t$ with C_k^2 and $2t$ respectively).

LA3: Consider $J, J' \in \mathcal{V}_t^n$. We have to show that if $P1^{prv}(J) \wedge \exists I, I' : J \leq I \wedge J' \leq I' \wedge dist(I, I') \leq t$, then $F^{prv}(J) = F^{prv}(J')$.

Since $P1^{prv}(J)$ holds, $\#_m(J) > 3t$ and $F^{prv}(J) = m$ also hold. From $I \geq J$, it follows that $\#_m(I) > 3t$. Then, since $dist(I, I') \leq t$, I' can differ from I in at most t entries and those different entries may contain m in I. Hence, $\#_m(I') \geq \#_m(I) - t$. Thus we obtain $\#_m(I') > 2t$. From $J' \in \mathcal{V}_t^n$ and $I' \geq J'$, it follows that at most t entries of J', which are occupied by value m in I', can contain the default value. Therefore, $\#_m(J') \geq \#_m(I') - t$. It implies that $\#_m(J') > t$, and hence $F^{prv}(J') = m = F^{prv}(J)$.

LA4: Consider $J, J' \in \mathcal{V}_t^n$. We have to show that if $P2^{prv}(J) \wedge \exists I : J \leq I \wedge J' \leq I$, then $F^{prv}(J) = F^{prv}(J')$.

Since $P2^{prv}(J)$, it follows that $\#_m(J) > 2t$ and $F^{prv}(J) = m$ holds. From $I \geq J$, it is implied that $\#_m(I) > 2t$. Then, as $J' \leq I$ and $J' \in \mathcal{V}_t^n$, at most t entries of J', which are occupied by m in I, can contain the $\perp$ value. Therefore, $\#_m(J') \geq \#_m(I) - t$. and we thus obtain $\#_m(J') > t$. It follows that $F^{prv}(J') = m = F^{prv}(J)$.

LU5: This property is trivially satisfied because $F^{prv}(J)$ is either m (when $\#_m(J) > t$) or the most frequent non default value in J. □

4 Algorithm *DEX*

In this section, we present a generic doubly-expedited algorithm *DEX* for one-step Byzantine consensus that can be instantiated with any legal condition-sequence pair.

Figure 1 provides the pseudocode of the algorithm. It uses an extra communication mechanism, called *Identical Broadcast*, that corresponds to the primitives Id-Send() and Id-Receive(). In contrast, P-Send() and P-Receive() correspond to the standard send/receive primitives. The underlying consensus is served by two primitives UC_propose(v) and UC_decide(v) which correspond to proposal of a value v and decision by v respectively.

Informally, the identical broadcast guarantees the delivery of the same message to all processes, even if the message is sent by a faulty process. Figure 2 shows how the Identical broadcast works. Its formal specification is described as follows:

```
Function $Consensus(v_i)$

init: $J1_i, J2_i \leftarrow \perp^n$ , $decided_i \leftarrow$ False , $proposed_i \leftarrow$ False

begin
     1 :   Upon Propose($v_i$) do:
     2 :       $J1_i[i] \leftarrow v_i$ ; $J2_i[i] \leftarrow v_i$
     3 :       P-Send($v_i$) to all processes;
     4 :       Id-Send($v_i$) to all processes;

     5 :   Upon P-Receive($v_j$) from any process $p_j$ do:
     6 :       $J1_i[j] \leftarrow v_j$;
     7 :       if $|J1_i| \geq n - t$ and P1($J1_i$) and $decided_i$ = False then
     8 :           Decide$_i$($F(J1_i)$); $decided_i$ = True
     9 :       end if

    10 :   Upon Id-Receive($v_j$) from any process $p_j$ do:
    11 :       $J2_i[j] \leftarrow v_j$;
    12 :       if  $|J2_i| \geq n - t$ and $proposed_i$ = False then
    13 :           UC_propose ($F(J2_i)$);
    14 :           $proposed_i$ = True;
    15 :       end if
    16 :       if $|J2_i| \geq n - t$ and P2($J2_i$) and $decided_i$ = False then
    17 :           Decide$_i$($F(J2_i)$); $decided_i$ = True
    18 :       end if

    19 :   Upon UC_decide($v$) do:
    20 :       if $decided_i$ = False then
    21 :           Decide$_i$($v$) ; $decided_i$ = True
    22 :       end if
end
```

Figure 1: Algorithm DEX: Doubly-Expedited Adaptive algorithm for Byzantine Consensus

Termination If a correct process invokes Id-Send(m), Id-Receive(m) occurs on all correct processes.

Agreement If two correct processes invoke Id-Receive(m_1) and Id-Receive(m_2) for the same sender, $m_1 = m_2$ holds.

Validity For any sender p_j, a correct process p_i invokes Id-Receive(m) exactly once and only if p_j invokes Id-Send(m).

Notice that the use of the identical broadcast does not introduce any additional assumptions to the system. This identical broadcast can be implemented just by using only the standard send/receive primitives. The implementation is easily obtained as a weaker form of simulating identical Byzantine failure model on the top of general Byzantine failure models[1]. The implementation of identical broadcast is given in the appendix. It should be noted that, in that implementation, a single communication step of the identical broadcast is realized by two communications steps of standard send/receive primitives. In our algorithm, the identical broadcast is used to develop the two-step decision scheme. In that sense, our two-step decision scheme can be regarded as a one-step decision scheme in identical Broadcast system.

In our algorithm, the part made up of lines 5-9 corresponds to one-step decision, and the another one made up of lines 10-18 corresponds to two-step decision. The algorithm works as follows: Each process

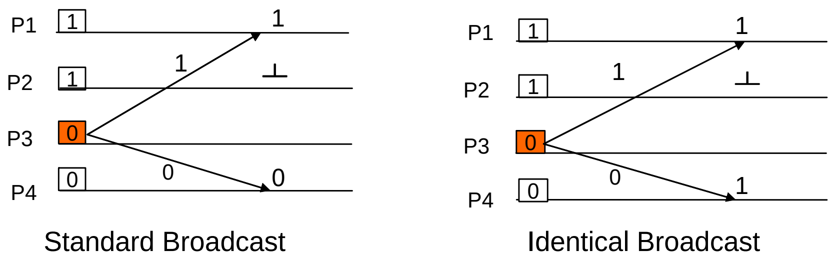

Figure 2: How Identical broadcast works: Let $P1, P2, P4$ are correct and $P3$ is faulty; Even if $P3$ sends different messages to $P1$ and $P4$, they receive the same message.

p_i starts a consensus execution with the invocation of *Consensus(v_i)* where v_i is its initial proposal value. The process p_i sends v_i to other processes by using both *P-send()* and *Id-send()* concurrently, and waits for receiving messages from other processes. By receiving messages, each p_i constructs two views $J1_i$ and $J2_i$, which correspond to one- and two-step decisions respectively. The views $J1_i$ and $J2_i$ are maintained incrementally. That is, they are updated at the reception of each message. When at least $n-t$ messages are received at $J1_i$, p_i tries to make a decision by evaluating $P1(J1_i)$. If $P1(J1_i)$ is true, p_i immediately decides $F(J1_i)$, that is, it decides in one-step. Otherwise, p_i continues to update $J1_i$. Similarly, when p_i receives at least $n-t$ messages at $J2_i$, it activates the underlying consensus with $F(J2_i)$. In addition, p_i evaluates $P2(J2_i)$ to check whether $J2_i$ is sufficient for taking decision. If $P2(J2_i)$ is true, p_i immediately decides $F(J2_i)$, that is, it decides in two steps. Otherwise, p_i repeats the check with each update at $J2_i$. Also, when the underlying consensus decides, each p_i simply borrows the decision of the underlying consensus unless it has decided already.

Notice that, unlike the existing Byzantine algorithms, DEX allows the processes to collect messages from all correct processes. This is the real secret of its ability to provide fast termination for more number of inputs.

4.0.1 Correctness.

We prove the correctness of our algorithm by showing that it provides one-step or two-step decision when it is instantiated with any legal condition-sequence pair (S^1, S^2). In the following proofs, let I be the actual input vector and I_i^1, I_i^2 be vectors obtained respectively from the views $J1_i$, $J2_i$ by replacing the default values with corresponding values in I.

Lemma 1 (Termination) Each correct process p_i eventually decides.

Proof Since there are at most t Byzantine processes, each correct process p_i receives messages from at least $n-t$ processes. It implies that, at some point $|J2_i| \geq n-t$. Hence, p_i certainly initiates the underlying consensus. Since the underlying consensus guarantees termination, p_i can decide when the underlying consensus decides. It follows that each process eventually decides.

Lemma 2 (Agreement) No two correct processes decide different values.

Proof Let two correct processes p_i and p_j decide v_i and v_j respectively. Then, we prove that $v_i = v_j$. Consider the following six cases.

- **(Case 1:)** *When both p_i and p_j decide in one step at line 8.*

  Since both p_i and p_j decide in one step, $P1(J1_i)$ and $P1(J1_j)$ hold. Let us consider two vectors I_i^1, I_j^1. From the definition of I_i^1, I_j^1, it follows that $J1_i \leq I_i^1$ and $J1_j \leq I_j^1$ hold. Since there are at most t Byzantine processes and only the Byzantine processes send different values to distinct processes, the vectors I_i^1, I_j^2 can differ in at most t entries . Hence, $dist(I_i^1, I_j^1) \leq t$ also holds. From property *LA3*, it is clear that $v_i = F(J1_i) = F(J1_j) = v_j$. Thus, we can conclude that p_i and p_j decide the same value.

- **(Case 2:)** *When p_i decides in one step at line 8 and p_j decides in two steps at line 17.*

  Let p_i and p_j decide in one and two step(s) using $J1_i$ and $J2_j$ respectively. This implies that $P1(J1_i)$ and $P2(J2_j)$ hold. Similar to Case 1, let us consider two vectors I_i^1, I_j^2. From the definition of I_i^1, I_j^2, it follows that $J1_i \leq I_i^1$ and $J2_j \leq I_j^2$. Since each correct process broadcasts the same value to all processes using P-send() and Id-send(), the vectors I_i^1, I_j^2 can differ only in Byzantine entries. As there are at most t Byzantine processes, $dist(I_i^1, I_j^2) \leq t$ holds. From property *LA3*, it is clear that $v_i = F(J1_i) = F(J2_j) = v_j$. Since p_j decides using $J2_j$, its decision value is v_i.

- **(Case 3:)** *When both p_i and p_j decide in two steps at line 17.*

  Since p_i and p_j decide in two steps, $P2(J2_i)$ and $P2(J2_j)$ hold. From the agreement property of the identical broadcast, it follows that if an entry in $J2_i(J2_j)$ contains a non-default value v, then the same entry in $J2_j(J2_i)$ contains either v or $\perp$. Hence, it is possible to have an vector I' such that $\forall k(1 \leq k \leq n) : (J2_i[k] \neq \perp \Rightarrow I'[k] = J2_i[k]) \wedge (J2_j[k] \neq \perp \Rightarrow I'[k] = J2_j[k])$. This implies that $J2_i \leq I'$ and $J2_j \leq I'$ hold. From property $LA4$, we thus get $v_i = F(J2_i) = F(J2_j) = v_j$.

- **(Case 4:)** *When p_i decides in one step at line 8 and p_j decides using underlying consensus at line 21.*

  Since p_j decides using the underlying consensus, and the underlying consensus satisfies unanimity, it is sufficient to show that every correct process p_k proposes v_i at line 13. We know that p_i decides using $J1_i$ and p_k uses $J2_k$ to propose a value to the underlying consensus. Consider the two vectors I_i^1, I_k^2. By using the same argument in case 2, we can show that $J1_i \leq I_i^1$, $J2_k \leq I_k^2$ and $dist(I_i^1, I_k^2) \leq t$ hold. Then, from property *LA3*, we can get $v_i = F(J1_i) = F(J2_k) = v_k$. It implies that each process p_k proposes v_i.

- **(Case 5:)** *When p_i decides in two steps at line 17 and p_j decides using underlying consensus at line 21.*

  Since p_j decides by the underlying consensus, similar to Case 4, we have to show that every correct process p_k proposes v_i to the underlying consensus at line 13. We know that p_i decides using $J2_i$ and p_k uses $J2_k$ to propose a value to the underlying consensus. By using the same argument in case 3, we can prove that there exist an vector I' such that $J2_i \leq I'$ and $J2_k \leq I'$. Then, from property $LA4$, it is clear that $v_i = F(J2_i) = F(J2_k) = v_k$. Hence, we can conclude that each process p_k proposes v_i.

- **(Case 6:)** *When both p_i and p_j decide at line 21:* Since the underlying consensus guarantees agreement property, we can conclude that $v_i = v_j$.

Lemma 3 (Unanimity) If all correct processes propose the same value v, then no correct process decides a value different from v.

Proof Let f be the actual number of Byzantine processes, and all correct processes propose the same value v. Since $f \leq t$, at each correct process p_i, no value except v appears more than t times in $J1_i$ and $J2_i$. If p_i decides at line 8 or 17, its decision value is either $F(J1_i)$ or $F(J2_i)$. From the definition of *LU5*, it follows that $F(J1_i) = F(J2_i) = v$. Hence, p_i decides v. In addition, since each p_i proposes $F(J2_i)$(that is, v) to the underlying consensus and the underlying consensus satisfies unanimity, any correct process that decides using underlying consensus decides only v. Hence, the unanimity holds.

Lemma 4 The algorithm *DEX* guarantees one-step decision for any input vector I, $I \in C_k^1$ if at most k processes exhibit Byzantine behavior.

Proof Since there are at most k Byzantine processes, each correct process p_i is guaranteed to receive messages from $n - k$ correct processes. Hence, eventually $dist(J1_i, I) \leq k$ holds. From property $LT1$, it follows that p_i decides in one step.

Lemma 5 The algorithm *DEX* guarantees two-step decision if the input vector I belongs to C_k^2 and at most k processes are Byzantine.

Proof As stated in lemma 4, since there are at most k Byzantine processes each correct process p_i receives messages from all $(n - k)$ correct processes. Hence, eventually $dist(J2_i, I) \leq k$ holds. From property $LT2$, it is clear that p_i decides in two steps.

The above lemmas imply the following theorem:

Theorem 3 For any instantiation with legal condition-sequence pairs, the algorithm *DEX* is a doubly-expedited one-step consensus algorithm.

5 Conclusion

Typically, Byzantine consensus algorithms guarantee one-step decision only in favorable situations, and no one-step algorithm supports two-step decision. In this paper, we proposed a novel one-step Byzantine algorithm *DEX* to circumvent these impossibilities. *DEX* has two distinguished features: Adaptiveness and double-expedition property. Due to adaptiveness, its conditions are sensitive only to the actual number of failures, and hence it achieves fast termination for more number of inputs when there are fewer failures. In addition, the double-expedition property enables it to support two-step decision in addition to one-step decision. Even though *DEX* takes four steps at worst in well-behaved runs while existing algorithms takes only three, it provides fast termination for more number of inputs. Practically, this is a favorable feature because the worst case does not occur so often in real systems, which makes us to expect that our algorithm can work efficiently on the average.

References

[1] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics.* Wiley, 2004.

[2] V. Brasileiro, F. Greve, A. Mostéfaoui, and M.Raynal. Consensus in one communication step. In *Proc. of the 6th International Conference on Parallel Computing Technologies*, volume 2127 of LNCS.

[3] D. Dobre and N. Suri. One-step consensus with zero-degradation. In *Proc. of the International Conference on Dependable Systems and Networks(DSN'06)*, pages 137–146, 2006.

[4] P. Dutta and R. Guerraoui. Fast indulgent consensus with zero degradation. In *Proc. of the 4th European Dependable Computing Conference on Dependable Computing*, volume volume 2485 of LNCS, pages 191–208, London, UK, 2002. Springer-Verlag.

[5] R. Friedman, A. Mostefaoui, and M. Raynal. Simple and efficient oracle-based consensus protocols for asynchronous byzantine systems. *IEEE Transactions on Dependable and Secure Computing*, 2(1):46–56, 2005.

[6] R. Guerraoui and M. Raynal. The information structure of indulgent consensus. *IEEE Transactions on Computers*, 53(4):453–466, 2004.

[7] T. Izumi and T. Masuzawa. Condition adaptation in synchronous consensus. *IEEE Transactions on Computers*, 55(7):843–853, 2006.

[8] T. Izumi and T. Masuzawa. One-step consensus solvability. In *Proc. of the 22nd international symposium on Distributed Computing(DISC'06)*, volume 4167 of LNCS, pages 224–237. Springer, 2006.

[9] I. Keider and S. Rajsbaum. On the cost of fault-tolerant consensus when there are no faults. *SIGACT News*, 32(2):45–63.

[10] A. Mostefaoui, S. Rajsbaum, and M. Raynal. Conditions on input vectors for consensus solvability in asynchronous distributed systems. In *Proc. of the thirty-third annual ACM symposium on Theory of computing (STOC'01)*, pages 153–162, 2001.

[11] A. Mostéfaoui, S. Rajsbaum, and M. Raynal. Using conditions to expedite consensus in synchronous distributed systems. In *Proc. of the 17th international symposium on Distributed Computing(DISC'03)*, volume 2848 of LNCS, pages 249–263, 2003.

[12] Y. Song and R. Renesse. Bosco: One-step byzantine asynchronous consensus. In *Proc. of the 22nd international symposium on Distributed Computing(DISC'08)*, volume 5218 of LNCS.

A An Implementation of Identical Broadcast

This section presents an implementation of the Identical Broadcast communication system that helps to build our two-step decision scheme. The basic idea of identical Broadcast is that even if Byzantine processes send arbitrary messages, all processes that receive a message from a faulty process receive the same message. In identical broadcast system, to successfully broadcast a message, a process has to obtain a set of witnesses for this broadcast. Likewise, a correct process accepts a message only when it knows there are enough witnesses for this broadcast.

The pseudocode appears in Figure 3. As specified in section 4, Id-send() and Id-Receive() are the communication primitives of identical broadcast, and P-send() and P-receive() correspond to standard broadcast.

```
Code for $p_i$:
    init: $num \leftarrow 0$

    Upon Id-send$_i(m)$ do :
        P-send$_i(init, m)$ to all processes.

    Upon P-Receive$_i(init, m')$ from $p_j$ do :
        if first-echo$(j)$ then
            P-send$_i(echo, m', j)$ to all processes.

    Upon P-Receive$_i$ $(echo, m', j)$ do :
        $num$ = number of copies of $(echo, m', j)$
            received so far from distinct processes.
        if $num \geq n - 2t$ and first-echo$(j)$ then
            P-send$_i(echo, m', j)$ to all processes.
        if $num \geq n - t$ and first-accept$(j)$ then
            Id-Receive$_i(m')$.
```

Figure 3: Algorithm IDB: An algorithm for Identical Broadcast.

To broadcast (that is, Id-send()) a message m, each process p_i P-Sends$(init, m)$ to all processes. When p_i P-Receives a first $(init, m')$ message from a process p_j, it act as a witness for that broadcast and P-sends its own message $(echo, m', j)$ to all processes. Also, when it collects at least $n - 2t$ same echo messages, it becomes a witness for that message and sends its own echo message to all processes.

When p_i P-Receives at least $n-t$ same echo messages, it accepts that message by invoking Id-receive() if it has not already accepted a message from p_i.

When p_i invokes the function first-accept(j), it returns true if and only if p_i has not already accepted a message for p_j. Similarly, when the function first-echo(j) is invoked, it returns true if and only if p_i has not sent a echo message for p_j. From the code, it directly follows that two rounds of standard broadcast is required to construct each round of identical broadcast.

We now prove correctness of the algorithm **IDB**.

Theorem 4 *Let $n > 4t$. Algorithm* IDB *implements identical broadcast system.*

Proof **Termination :** Let a correct process p_i Id-Sends a message m. To do that, p_i P-Sends$(init, m)$ to all processes. Consequently, each correct process p_j P-Receives$(init, m)$, and then it P-sends$(echo, m, i)$ message to all processes. As there are at most t Byzantine processes, each p_j eventually P-Receives at least $n-t$ $(echo, m, i)$ messages. As a result, Id-Receive(m) occurs at all correct processes.

Agreement: Proof by contradiction. Let two correct processes p_i, p_j invoke Id-receive(m), Id-receive(m') for a process p_h such that $m \neq m'$. It implies that p_i and p_j must have collected $n-t$ echo messages from distinct processes for m and m' each. Since $n > 4t$, these two sets share more than $2t$ common senders. Given that there are only t Byzantine processes, more than t of these common senders are correct processes. The function first-echo() ensures that any correct process P-Sends only a single echo message for p_h to all processes. Hence, $m = m'$. This is a contradiction.

Validity Assume a correct process p_i Id-Receives(m) from a process p_j. The function first-accept() ensures that p_i Id-Receives(m) exactly once for p_j. Since p_i collects at least $n-t$ $(echo, m, j)$ messages, at least $n-2t$ of them are from correct processes. A correct process P-sends $(echo, m, j)$ if it collects at least $n-2t$ $(echo, m, j)$ messages from distinct processes. Since $n-2t > t$, a correct process, that has sent an echo $(echo, m, j)$ message, must have received $(init, m)$ message from p_j. It implies that p_j must have sent m.

Thus, the algorithm IDB implements identical broadcast system. $\square$