Abstract—Renaming is a fundamental problem in distributed computing and consists in having a set of processes with unique ids from a large namespace pick distinct names from a smaller namespace. In a stronger, order-preserving variant of the renaming problem, new names are required to preserve the ordering of the initial ids.

This paper addresses order-preserving renaming in synchronous message passing systems with Byzantine failures. Although in this model, order-preserving renaming can be solved by using consensus, it is known that this problem is “weaker” than consensus. Therefore, we are interested in designing more efficient algorithms than a consensus-based solution and make three contributions in this direction. We present an order-preserving renaming algorithm with $N > 3t$ resiliency, target namespace of size $N + t - 1$, and $O(\log N)$ round complexity ($N$ is the number of processes and $t$ is an upper bound on the number of faults). Similarly to the existing crash-tolerant solution [13], our algorithm is based on approximate agreement. We also show that with $N > t^2 + 2t$ our algorithm can be modified to have constant round complexity while achieving tight namespace of size $N$. Finally, we present an algorithm that solves order-preserving renaming in just 2 communication rounds with $N > 2t^2 + t$.

I. INTRODUCTION

Renaming is a fundamental problem in distributed computing, which can be informally described as follows: a set of processes $\{p_1, \ldots, p_N\}$ with unique ids in the range $[1 \cdots N_{\text{max}}]$ must pick new names from a given range $[1, \ldots, M]$, where $M \ll N_{\text{max}}$. The range of values to which new names belong is called target namespace. In this work, we are interested in order-preserving variant of the renaming problem that requires processes to preserve the order of their initial ids. This variant is interesting as it allows to use renaming in settings where the original identifiers encode some additional information, such as, for instance, their relative priority in accessing a shared resource.

In this paper we assume a synchronous message-passing model. Although in this model order-preserving renaming can be solved by using consensus, it is known that this problem is “weaker” than consensus, therefore we are mainly interested in solutions that do not require solving consensus. In synchronous message passing, order-preserving renaming has only been previously addressed in the context of crashes [13]. The algorithm in [13] uses approximate agreement to solve renaming. The approximate agreement problem (or AA for short), introduced in [6], requires processes, starting with arbitrary real values, output values within some bounded distance from each other.

The order-preserving algorithm of [13] only tolerates crashes, and adapting previous work to cope with Byzantine processes raises several interesting challenges. The first challenge is that original identifiers are not globally known among the processes a priori. Note that with this knowledge it becomes trivial to solve order-preserving renaming without any communication (just by sorting the set of ids and then choosing the rank of each id as new name). Therefore, the usual techniques of translating a crash-tolerant algorithm into a Byzantine-tolerant algorithm [3], [12], which assume that the identifiers are known a priori to each process, cannot be directly applied in our setting. We could further explore the possibility of adapting the translation techniques to dealing with the unknown ids and then apply them to the crash-tolerant algorithm of [13]. In fact, the authors of [14] took this approach in designing the first Byzantine-tolerant (non order-preserving) renaming algorithm. The downside of this approach is that the translation techniques increase the round complexity by the factor of 4 and introduce a linear increase in the message complexity because processes must broadcast and echo histories of previously received messages.

In contrast, our first Byzantine-tolerant algorithm follows the structure of the crash-tolerant algorithm of [13] with some additional adaptations that do not yield any increase in the message and round complexity.

Our algorithm consists of two phases. The purpose of the first phase is to exchange the information about the ids existing in the system. This is achieved by a 3-round id selection scheme that restricts the number of ids in the system, despite lies by Byzantine processes. Our id selection is conceptually similar to the Gradecast [8] primitive which assigns confidence levels to messages delivered from other processes.

In the second phase, each process proposes a new name for each known id based on the rank of this id in the ordered set of all identifiers known to this process. The proposed names are then used as inputs into separate instances of Byzantine-tolerant approximate agreement, one per id. Finally, each process returns as its new name the output of the corresponding instance of AA.

The Byzantine-tolerant AA [6] guarantees that the outputs are within the range of values issued by the correct processes. However, since Byzantine processes can cause correct processes to propose overlapping intervals of values for different instances of AA, without any additional care, the outputs still may not preserve the initial ordering. We solve this by
simultaneously validating messages from the same process into all concurrent instances of AA. Interestingly, this validation does not require any additional messages.

We then analyze the resulting algorithm when \( N \) is large compared to \( t \). In the lines of the work for crash-faults reported in [1], we show that the AA-based approximation phase, and thus our algorithm, requires only a constant number of rounds to converge when \( N > t^2 + 2t \). Furthermore, in this case it also achieves tight namespace of size \( N \), because our id selection scheme ensures that Byzantine processes are not able to introduce more than \( t \) identifiers.

Even in the favorable case above, the number of communication rounds can be an impairment for time constrained applications. Therefore, we then address the challenge of performing order-preserving renaming in as few communication rounds as possible. We show that, if the number of faults is restricted to \( N > 2t^2 + t \), it can be solved in just 2 rounds. This is done by having processes exchange their initial ids, perform one echoing round, and then count the numbers of echoes to calculate new names.

**Contributions**

To our knowledge, our work is the first to address order-preserving renaming with Byzantine faults. Our results also improve the existing work on non order-preserving renaming.

Our first contribution is an algorithm that performs order-preserving renaming with \( N > 3t \) resiliency and \( O(\log N) \) round complexity. Additionally, our algorithm presents an improvement on the namespace size, \( N + t - 1 \), compared to the previous result of \( 2N \) [14].

Our second contribution is to show that, if \( N > t^2 + 2t \) our algorithm can be modified to terminate in constant number of rounds and, interestingly, achieves in this case optimal namespace of size \( N \).

Our last contribution consists in a fast algorithm for \( N > 2t^2 + t \) that employs just 2 communication rounds and achieves the target namespace of size \( N^2 \).

All our algorithms are deterministic.

**Paper Organization**

The remainder of this paper is organized as follows. In Section II we introduce the system model and formally define the problem addressed in this paper. In Section III we discuss the existing work. Section IV is dedicated to the order-preserving renaming algorithm with \( N > 3t \). In Section V, we show that if \( N > t^2 + 2t \), the algorithm can be modified to terminate within a constant the number of rounds. In Section VI, we present a fast 2-round renaming algorithm. Finally, Section VII presents the conclusions and outlines the directions for future work.

**II. MODEL AND PROBLEM DEFINITION**

Processes are arranged in a synchronous fully connected network of an a priori known size \( N \). Each pair of processes communicates by message passing through a direct communication link. Communication channels are assumed to be reliable.

The links of each process are labeled by \( 1, \ldots, N \), where links \( 1, \ldots, N - 1 \) are to the remaining processes and link \( N \) is a self-loop. It is assumed that processes know the label of the link through which any message is received.

Each correct process has a unique identifier originally only known to the process itself. Up to \( t \) processes may be faulty and exhibit arbitrary behavior – these processes are named Byzantine.

The renaming problem can be formally defined by the following conditions [2]:

- **Validity:** Each new name is an integer in the range \([1 \ldots M]\).
- **Termination:** Each correct process outputs a new name.
- **Uniqueness:** No two correct processes output the same new name.

The particular case in which the size of the target namespace is equal to \( N \) is called strong renaming.

In this paper we are interested in order-preserving variant of renaming, in which the uniqueness property is substituted by the following property:

- **Order-preserving** [2]: New names of correct processes preserve the order imposed by their original ids.

**III. RELATED WORK**

The renaming problem and its order-preserving variant were originally introduced by Attiya, Bar-Noy, Dolev, Peleg, and Reischuk [2] in the asynchronous message-passing model with crashes where consensus is known to be impossible [9]. The authors of [2] presented a renaming algorithm with \( N > 2t \) and target namespace of size \( N + t \), and an order-preserving renaming algorithm with \( N > 2t \) and target namespace of size \( 2'(N - t + 1) - 1 \). Both algorithms tolerate the optimal number of crashes and the bounds on the target namespace were also shown to be optimal [2], [10]. Since its introduction in [2], the renaming problem has been extensively studied in both message passing and shared memory. From this point, we limit our discussion to the results in the synchronous message passing model considered in this paper.

In synchronous systems, renaming can be solved using Reliable Broadcast [4] or consensus [11]: processes can simply agree on the set of ids existing in the system. Unfortunately, this approach requires linear round complexity [7], while renaming can be implemented in \( O(\log N) \) communication rounds [5] in this model. In fact, renaming is considered the simplest non-trivial distributed computing task [5]. It is therefore no surprise that a significant research effort has been placed in devising efficient algorithms for the renaming problem.

A crash-tolerant algorithm that implements strong renaming was proposed in [5] and works as follows. A process chooses a new name by selecting one bit at a time, starting with the high-order bit and working down to the low-order bit. In each round processes exchange their ids and the intervals of new
names in which they are interested. Then, processes split the received ids in two sets, choosing 0 if their own id belongs to the first half, or 1 otherwise, and repeat the procedure. The round complexity of the algorithm is $O(\log N)$ which was also shown to be optimal if $t = N - 1$ [5].

A crash-tolerant algorithm that implements strong order-preserving renaming was presented in [13]. As discussed earlier, the algorithm of [13] uses the approximate agreement primitive (AA): processes exchange their old ids, propose a new name for each id based on this id’s rank in the list of all identifiers known to the given process. Due to crashes, processes may have received different sets of identifiers and therefore may propose different names for the same id. These discrepancies are later reduced by separate instances of AA, each one per identifier. After the approximate agreement brings the values within safe distance from each other, each process decides on a new name based on the output of the corresponding instance of AA. The round complexity of the algorithm is $O(\log N)$. Recently, the authors of [1] made the algorithm of [13] early deciding, i.e. the time complexity of the algorithm depends on $f$, the number of actual faults occurred in a given run. The complexity of the early-deciding version of the algorithm is $O(\log f)$. Interestingly, the authors observed that the algorithm can decide in constant number of rounds if the number of actual faults is bounded by $N > 2f^2$. This is because in that case approximate agreement converges in a constant number of iterations.

Byzantine renaming in message-passing systems has been addressed in [14], where the authors prove the lower bound of $N > 3t$ on the number of Byzantine failures for renaming algorithms with bounded time complexity (i.e., the complexity depends only on the size of the system). The paper also proposes a non order-preserving renaming algorithm, adapting the automatic crash-to-Byzantine translation techniques introduced in [3], [12] to the crash-tolerant algorithm of [5]. The main difficulty in adapting the translation techniques of [3], [12] consists in eliminating the assumption that all initial ids are known to each process a priori. Since Byzantine processes can announce different identifiers that the correct processes are not able to recognize as faulty, in the resulting transformed algorithm, the target namespace is increased to $2N$. The algorithm tolerates $N > 3t$ Byzantine failures and runs in $O(\log N)$ rounds.

IV. ORDER-PRESERVING BYZANTINE RENAMING

In this section, we present what is, to our knowledge, the first order-preserving renaming algorithm tolerant to Byzantine faults. The resiliency of our algorithm is $N > 3t$. Semantically, our algorithm follows the structure of the order-preserving algorithm for the fail-stop model presented in [13] employing the techniques of Byzantine approximate agreement (AA) introduced in [6] with extensions that address two additional concerns. First, we limit the number of identifiers introduced by the faulty processes. Second, we ensure that, in spite of contradictory information sent by Byzantine participants, the instances of AA converge in a consistent way ensuring that new names preserve initial ordering.

The algorithm, depicted in Alg. 1, uses two distinct phases, namely the id selection phase and the rank approximation phase, or voting. The first phase takes 3 rounds and aims at limiting the number of identifiers produced by faulty processes while ensuring that all correct processes know all correct ids. At the end of this phase, each process makes an estimate of the new name for each process. Since these estimates are not precise enough to be order-preserving, the second phase of the algorithm runs, in parallel, coordinated Byzantine-tolerant approximate agreements on those estimates. This phase is called approximation phase and takes a logarithmic number of rounds. We denote each round of the approximation phase as a voting round. By making appropriate validations on the votes of each process, we ensure that the values converge preserving the order of original ids. In the following subsections we discuss each of the two phases in detail.

A. Id Selection Phase

This phase is implemented in ROUNDS 1 to 3 of Alg. 1. The purpose of the id selection phase is to choose which identifiers should feed the rank approximation phase. Note that Byzantine processes can announce different ids to different peers; if their power is not constrained the number of “fake” ids may prevent correct processes from executing correctly. On the other end, the purpose of this phase is not to ensure that all
correct processes select the exact same set of identifiers: that would be equivalent to solving consensus, which would have linear round complexity. For convenience of exposition, ids belonging to correct processes are named correct ids. All other ids are referred to as Byzantine, e.g. ids issued by Byzantine processes as their own or non-existent ids that Byzantine processes claim to have received from others.

Each process locally stores the following variables: two sets, timely and accepted, that are used to collect ids; variable ranks, a sparse array where ranks[id] stores a new name for each id in the set set. Function SORT(set) orders the entries in a set set; function RANK(set, v) returns a position of value v in the ordered set set.

At the end of the id selection phase, the following properties are ensured on the timely and accepted sets:

- at every correct p, timely_p includes all correct ids;
- at every correct p, accepted_p includes at most N + t − 1 ids in total;
- at every correct p, accepted_p is such that: \( \bigcup_{q: q \text{ is correct}} \text{timely}_q \subseteq \text{accepted}_p \),

i.e., if one id is considered timely by some correct process, this id is for sure included in the accepted set by every other correct process (but not necessarily considered timely).

In detail, this phase of the algorithm works as follows. In Round 1, each correct process broadcasts its identifier in an IO message. In Round 2, processes echo the ids they have received in the previous round (ECHO messages). Only ids that have been echoed at least N − t times are considered for the following round. This effectively limits the number of Byzantine ids. Also, since all correct ids are echoed by the correct processes, all correct ids are taken to the next round. Ids that satisfy the previous condition are broadcast in a READY message in Round 3; all ids for which at least \( N − t \) READY messages have been issued are added to the timely set. Notice that all correct ids will be included in the timely set of every correct process. All ids for which at least \( N − 2t \) READY messages have been produced are added to the accepted set. As a result, accepted also contains all ids in the timely set.

The concept of separating the ids in timely and accepted sets is similar to grading the delivered messages with confidence levels, as done in Gradecast. The classical broadcast [4] and Gradecast [8] algorithms require each process to know the identity of a sender. Therefore, if the ids are not known a priori and all processes are broadcasting at the same time, Byzantine participants can collude such that more than \( t \) messages issued by Byzantine processes are delivered by the correct processes. In fact, any message received in the first round by at least \( N − 2t \) correct processes can be delivered by a correct process. Therefore, in our id selection, the size of the accepted set at a correct process can contain as many as \( N + t − 1 \) ids. Note also that Byzantine processes may use correct ids as their own; this has no effect on the execution: duplicate identifiers do not appear in timely and accepted sets.

### Algorithm 2 Procedure isVALID

01 Function isVALID (timely, ranks) returns boolean is
02 foreach id, id’ ∈ timely such that id < id’ do
03 if id /∈ ranks or id’ /∈ ranks or ranks[id]−ranks[id’] < \( \delta \) then
04 return false;
05 return true;

### Algorithm 3 Procedure APPROXIMATE

01 Function APPROXIMATE (my_ranks, all_ranks) returns array of ranks is
02 new_ranks := \( \emptyset \);
03 foreach id ∈ accepted do
04 votes[id] := \( \emptyset \);
05 foreach R ∈ all_ranks do
06 if id ∈ R then
07 votes[id] := votes[id] \( \cup \) R[id];
08 accepted := \{id ∈ accepted : \|votes[id]\| \geq N − t\};
09 foreach id ∈ accepted do
10 for votes[id] + 1 to N do // fill missing votes with valid vote
11 votes[id] := votes[id] \( \cup \) my_ranks[id];
12 for 1 to t do // remove t extreme values
13 votes[id] := votes[id] \( \setminus \) \{MAX(votes[id]);
14 votes[id] := votes[id] \( \setminus \) \{MIN(votes[id]);
15 sort(votes[id]);
16 new_ranks[id] := AVG(sort(votes[id]));
17 return new_ranks;

At the end of the id selection phase, each process sorts its accepted set, and estimates a new name to each of these ids (including its own), which is the rank of that id in the sorted set stretched by factor \( \delta = 1 + \frac{1}{3(N + t)} \). This factor is large enough to prevent names from clashing due to small disagreement errors in the approximate agreement, as we explain below. The purpose of the second phase is to iteratively execute approximate agreement until the ranks calculated by the correct processes are within safe distance.

### B. Approximation Phase

The approximation phase, or voting, starts in Round 4 and takes logarithmic number of rounds to converge. This phase is based on the Byzantine-tolerant AA algorithm of [6]. The AA algorithm guarantees that, in spite of contradictory inputs from Byzantine processes, the output values are within a bounded error. Moreover, it guarantees that the outputs are within the range of input values issued by the correct processes. In our case, the ranks calculated in the id selection phase may not preserve the correct global ordering. As a result, the ranges of the correct inputs into AA may overlap. Without any additional care, AA may converge to values that are not order-preserving.

The above issue is addressed by the verification function depicted in Alg. 2 that aims at ensuring that the approximation is performed in accordance with the ordering of the original ids. The function ISVALID takes as input the timely set of a local process and array ranks received from some other process. It makes two tests to check if the votes from the remote process are consistent. First, the votes must include a vote for each id in timely (we remind that if \( p \) and \( q \) are correct, then \( \text{timely}_p \subseteq \text{accepted}_q \), thus any vote that does not satisfy this invariant may be discarded as faulty). Second, it ensures that the new rankings for these ids appear in the correct order.
separated by the minimum safety margin of $\delta$. As a result, if a Byzantine process sends different votes to different processes and both are considered valid, the presented validity conditions are sufficient to ensure that the approximation of the validated values will be done in a consistent way.

In addition to the variables and functions introduced before, the second phase of our algorithm also needs the following data structures and auxiliary functions: variable $R$ is a set of $ranks$ arrays; the function $ROUND(x)$ returns the integral value nearest to $x$; finally, the function $SELECT_k(set)$ returns a choice of values from a set. These values are chosen to maximize the convergence rate of the approximate agreement. Later in the text we describe what is the most appropriate choice function.

In detail, each voting round works as follows. Processes exchange the values in their $ranks$ array. Each array received from a remote process is first validated as described earlier. If the array is considered valid, it is added to the set of votes received in the current round. At the end of the round, votes are processed by the function $APPROXIMATE$, depicted in Alg. 3. In this function, each process computes a new rank for each id in the $accepted$ set as follows. It first collects all votes received for a given id into multiset votes[$id$] (multiset is a set that allows repetitions). If for some id in $accepted$, less than $N - t$ votes are received, this id is discarded (by construction, this never happens to an id that has been considered timely by some correct process). For the remaining ids, if the number of votes is less than $N$, the process fills the multiset by including copies of its own value (local values of a correct process are always valid). Then, the resulting multiset of $N$ votes is sorted and the $t$ lower values and $t$ higher values are discarded. Finally, function $SELECT_{t}$ is used to pick a subset from the remaining values that is averaged to compute the new vote for that id. This function returns a multiset consisting of each $(it + 1)th$ element of the set (which is an ordered multiset), where $0 \leq i < \lceil \frac{|set|}{t} \rceil$. In other words, $SELECT_{t}(set)$ returns a multiset consisting of the smallest and each $t$-th element after it. This choice of $SELECT_{t}$ is the same as in the approximate agreement algorithm of [6], which guarantees the convergence rate of $\sigma_t = \lceil \frac{N-2t}{t} \rceil + 1$ where $\sigma_t$ is a number of elements returned by $SELECT_{t}$.

After executing $2 \log t + 5$ approximation rounds, the new name is chosen as the rounded value of rank[$my_id$]. The stretch factor of $\delta$ applied to the inputs and the validation procedure ensures that the ranks converge preserving a distance of slightly more than 1, which prevents the rounded ranks from clashing in spite of a possible approximation error.

C. Correctness

Complete proofs are provided in Appendix A.

We start by stating that any id in $timely$ at some correct process, is necessarily included in $accepted$ of any other correct process.

Lemma 4.1: For any id such that id $\in timely_p$ at some correct $p$, then id $\in accepted_q$ at any correct $q$.

The following lemma states that all correct ids are included in $timely$ sets of all correct processes.

Lemma 4.2: If id belongs to some correct $p$, then id $\in timely_q$ at any correct $q$.

As discussed earlier, Byzantine processes can generate more than $t$ identifiers, none of which recognized as faulty by the correct processes. The following lemma bounds the total number of ids included in $accepted$ at any correct $p$.

Lemma 4.3: At the end of Round 3, if $p$ is correct, then $|accepted_p| \leq N + \lceil \frac{t^2}{N - 2t} \rceil$.

We then show that correct processes always issue valid votes.

Lemma 4.4: For any $r \geq 4$, if $ranks_p$ and $ranks_q$ are held by any two correct $p$ and $q$ in Round $r$, then $ISVALID(ranks_p, ranks_q) = true$.

Corollary 4.5: If id $\in timely_p$ at some correct $p$, then its rank is updated in every approximation round by each correct process.

Corollary 4.6: If id $< id'$ belong to two correct processes, then $ranks_p[id] + \delta \leq ranks_p[id']$, at any correct $p$ in every Round $r \geq 3$.

We now need to bound the maximum discrepancy in the initial ranks for the same ids.

Lemma 4.7: If id $\in timely_p$ for some correct $p$, then at the end of Round 3, $|ranks_p[id] - ranks_q[id]| \leq (t + \lceil \frac{t^2}{N - 2t} \rceil) \times \delta$, where $ranks_q[id]$ is the rank of id at some correct $q$.

Now it remains to show that each approximation round of Alg. 3 reduces the distance between the ranks by the factor $\sigma_t = \lceil \frac{N-2t}{t} \rceil + 1$.

Lemma 4.8: Let id $\in timely_p$ at some correct $p$, and $\Delta_r$ denote the maximum distance between the correct ranks for id in the beginning of Round $r$. Then, at the end of Round $r$, the distance between new correct ranks for this id is within the range of $\frac{\Delta_r}{\sigma_t}$. Moreover, the new values are within the range of the old values belonging to correct processes.

We now calculate the number of iterations needed to reduce $\Delta_r$ to less than $\frac{1-\delta}{2}$. $\Delta_4 \leq (2t-1) \times \delta$, then after $r = 2[\log t] + 5$ iterations, the range of the values belonging to all correct processes is less than $\Delta_r+4 < \frac{1-\delta}{2}$.

Finally, we are ready to prove the main theorem.

Theorem 4.10: Alg. 1 implements order-preserving renaming with $N > 3t$ and target namespace of size $N + t - 1$.

Proof:

Validity. By Lemma 4.3, if $p$ is correct, $|accepted_p| \leq N + \lceil \frac{t^2}{N - 2t} \rceil \leq N + t - 1$, for $N > 3t$. Therefore, the initial ranks are bounded by $(N + t - 1) \times \delta$. Since by Lemma 4.8, all correct processes output a value within the interval of the
initial correct values, the outputs of the correct processes are bounded by $\text{ROUND}((N + t - 1) \times \delta) = N + t - 1$.

**Termination.** After $2 \lceil \log t \rceil + 8$ rounds, every correct process outputs a value.

**Order-preserving.** By Lemmas 4.2, correct ids are always included in timely sets and, by Corollary 4.5, are updated in each round by every correct process. By Corollary 4.6, for any two correct id and id' such that id < id', the distance between their rankings is lower bounded by $\delta$ in every round. Since by Lemma 4.9, after $2 \lceil \log t \rceil + 8$ rounds, $\Delta_r < \frac{\delta - 1}{2}$,

$$\text{rank}(\text{id}) + \delta + \frac{1 - \delta}{2} < \text{rank}(\text{id'}) - \frac{1 - \delta}{2}.$$ 

Hence, $\text{ROUND}((\text{ranks}[\text{id}])) < \text{ROUND}((\text{ranks}[\text{id'}])$.

**D. Complexity Analysis**

The round complexity of Alg. 1 is $2 \lceil \log t \rceil + 8$. In each round, processes employ all-to-all communication. As a result, the total message complexity is $O(N^2 \log t)$. Since in each round processes exchange arrays of at most $N + t - 1$ original ids and their ranks, the message size is bounded by $O((N + t - 1) (\log N_{\max} + \log N))$ bits.

**V. Constant Time Renaming**

An interesting property of Alg. 1 is that it performs strong renaming, i.e. renaming with the target namespace of size $N$, and can terminate after a constant number of rounds if $N > t^2 + 2t$. The optimal namespace is achieved because Byzantine processes are not able to introduce any additional identifiers in our id selection scheme. The constant round complexity is due to the fast convergence of Byzantine AA. Similar argument was used by the authors of [1] to prove the constant round complexity of the crash-tolerant algorithm presented in [13] when the number of actual crashes is bounded by $N > 2f^2$.

This result is formalized below. Proofs are provided in Appendix B.

**Lemma 5.1:** If $N > t^2 + 2t$, Alg. 1 achieves the target namespace of size $N$.

**Lemma 5.2:** If $N > t^2 + 2t$, after 4 approximation rounds, the values held by the correct processes are within the distance of less than $\frac{\delta - 1}{2} = \frac{1}{6(N + 1)}$.

Therefore, if we change the code of Alg. 1 to run only 4 approximation rounds (Line 29), the resulting algorithm has the complexity of 7 rounds.

**Theorem 5.3:** The modified Alg. 1 implements strong order-preserving renaming in $O(1)$ rounds if $N > t^2 + 2t$.

**VI. 2-Round Renaming Algorithm**

In the previous section we have shown that Alg. 1 has constant round complexity with $N > t^2 + 2t$. This is an interesting result from the asymptotic point of view, specially considering that the resulting name space is optimal. Still, from the practical point of view, the number of communication rounds can still be an impairment for time constrained applications (the number of rounds of Alg. 1 is exactly 7). Therefore, in this section we are interested in performing renaming in as few communication rounds as possible. Interestingly, we show that order-preserving renaming in face of Byzantine faults can be solved in just 2 communication rounds with $N > 2t^2 + t$, by relaxing the target namespace to $N^2$. Obviously, in just 2 communication rounds, it is impossible to perform iterative approximate agreement. In fact, our algorithm is simply based on counting echoes that are filtered by a validity check.

The algorithm is depicted in Alg. 4. The main idea of the algorithm is having each process initially announce its ids to all other processes; then, echo all the ids received in the first round, and finally having each correct process calculate its new name by ordering all the received ids, and calculating offsets, i.e. spacings between two consecutive names, according to the number of echoes received for each id. Byzantine processes may opt not to echo the ids or even send contradictory information to different processes. Therefore, correct processes may receive different sets of ids as well as different numbers of echoes for each id. The key to the algorithm is to compute the offsets in such a way that the new names chosen by the correct processes will hold the order-preserving property, despite potentially inconsistent sets of echoes.

As the previous algorithms, Alg. 4 also uses a timely and an accepted set of ids. In this algorithm, all ids broadcast in Round 1 are considered timely and all ids echoed in Round 2, that pass a basic validity test, are accepted. The validity test, captured by function ISVALID, limits the power of Byzantine processes as follows: first, it only accepts echo messages from processes that have sent their id in Round 1; then, it does not accept a MULTIECHO message that has more than $N$ ids; finally, the incoming MULTIECHO must have at least $N - t$ ids in common with the timely set of the recipient (note that if the sender and the recipient of a MULTIECHO are correct, they both have at least $N - t$ correct ids in their timely sets). Also,
for each accepted id, the algorithm counts how many processes have echoed that id (again, correct ids are guaranteed to be echoed at least \( N - t \) times).

After all echo messages have been processed, processes are ready to calculate new names. The offset for each known id is simply the value of \( \text{MIN} (\text{counter}, N - t) \) (Line 20). The adjustment to \( N - t \) guarantees that these offsets for the correct ids are always the same. This prevents Byzantine processes from introducing an additional error linear in the number of correct processes by choosing to echo correct ids for some processes but not others. Finally, the new name of a process is produced by summing the offsets of all ids up to, and including, the id of the current process. The algorithm also stores locally estimated values of new names for other processes (Line 20); this is not required in practice and is done here only for clarity of the proofs.

A. Correctness

Proofs are provided in Appendix C.

Let \( \Delta \) denote the maximum possible discrepancy between the new names for some correct id.

**Lemma 6.1:** \( \Delta \leq 2t^2 \).

We now need to lower bound the offset of any correct id.

**Lemma 6.2:** Let \( \text{id} \) and \( \text{id}' \) be two correct identifiers. If \( \text{id}' < \text{id} \), then \( \text{newid}_p[\text{id}'] + (N - t) \leq \text{newid}_p[\text{id}] \) at some correct process. We are ready to prove the correctness.

**Theorem 6.3:** Alg. 4 implements order-preserving renaming with \( N > 2t^2 + t \) and the target namespace of size \( N^2 \).

**Proof:** Validity. The total number of echoed ids accepted by each correct process in Round 2 is at most \( N^2 \). Therefore, the correct processes output an integer value within the range \([1, \ldots, N^2]\). Hence, Alg. 4 satisfies the validity property.

**Termination.** After 2 rounds, every correct process outputs a value.

**Order-preserving.** Consider two correct processes \( p \) and \( q \) with initial identifiers \( \text{id} \) and \( \text{id}' \), such that \( \text{id} < \text{id}' \). By Lemma 6.2, \( \text{newid}_p[\text{id}'] + N - t \leq \text{newid}_p[\text{id}] \). Since by Lemma 6.1, \( \Delta \leq 2t^2 \), then \( \text{newid}_p[\text{id}'] - 2t^2 \leq \text{newid}_p[\text{id}'] \). Furthermore, since \( N > 2t^2 + t \), \( \text{newid}_p[\text{id}'] + N - t - 2t^2 < \text{newid}_p[\text{id}'] \).

B. Complexity Analysis

Alg. 4 consists of 2 communication rounds. Since in each round, processes employ all-to-all communication, the total message complexity is \( 2N^2 \). In Round 2, processes exchange vectors of all ids they have received in Round 1. Therefore, the message size is bounded by \( O(\sqrt{N}\log N_{\text{max}}) \) bits.

VII. Conclusions

This paper is the first to address order-preserving renaming in synchronous systems subject to Byzantine faults. Our contributions also improve the existing results on non-order-preserving renaming in this model.

Our first algorithm performs order-preserving renaming with \( N > 3t \) resiliency with the same time and message complexity of the existing crash-tolerant solution [13]. Additionally, our algorithm presents an improvement on the namespace size compared to the previous result of [14] and even achieves tight namespace with \( N > t^2 + 2t \). It remains open question whether it is possible to achieve tight namespace for other values of \( t \).

On the other hand, if the resiliency is bounded by \( O(\sqrt{N}) \), we have shown that order-preserving renaming can be performed in constant time both by using approximate agreement and with a simple echo-scheme. This resiliency threshold for renaming in constant time asymptotically matches the existing results for the crash-fault model [11]. Another open question is whether this threshold is optimal or better resiliency can be achieved in constant time.

Acknowledgments

The authors are thankful to Dan Alistarh and Corentin Travers for the discussion on an earlier version of this work. We also thank Marcos K. Aguilera and the anonymous referees for their helpful comments on the manuscript.

This work was partially supported by Fundação para a Ciência e Tecnologia (FCT) via the INESC-ID multi-annual funding through the PIDDAC Program fund grant, under project PEst-OE/ EEI/ LA0021/ 2011, and via the project PEPITA (PTDC/EEI-SCR/2776/2012).

References

APPENDIX A

Proof of Lemma 4.1: Assume by contradiction, \( id \notin accepted \) at some correct \( q \). This is only possible if \( q \) has not received \( N - 2t \) \( \langle \text{READY}, id \rangle \) messages in Rounds 3 (Lines 17-18 of Alg. 1). But if \( p \) added \( id \) into \( \text{timely} \), it means that it has received at least \( N - t \) \( \langle \text{READY}, id \rangle \) messages, \( N - 2t \) of which must have been sent by the correct processes in Round 3. Therefore all correct processes have received at least \( N - 2t \) \( \langle \text{READY}, id \rangle \) messages in Round 3, which leads to a contradiction.

Proof of Lemma 4.2: Assume by contradiction, \( id \notin timely \) for some correct \( q \). This means that \( q \) has not received \( N - t \) \( \langle \text{READY}, id \rangle \) in Round 3. This is only possible if some correct process has not issued \( \langle \text{READY}, id \rangle \), which in turn is because it has not received \( N - t \) \( \langle \text{ECHO}, id \rangle \) in Round 2. This also is only possible if \( id \) was not received by some correct process in Round 1. However, since \( p \) is correct, \( p \) sent \( id \) to all correct processes in Round 1. Contradiction.

The following lemma will be used to calculate the maximum number of ids that Byzantine processes are able to produce.

Lemma A.1: If \( id \in accepted \) at some correct \( p \), then at least \( N - 2t \) correct processes received \( id \) in Round 1.

Proof of Lemma A.1: If \( id \in accepted \), then \( p \) has received at least \( N - 2t \) \( \langle \text{READY}, id \rangle \) messages from which at least 1 must have been issued by a correct process. This means that some correct process received at least \( N - t \) \( \langle \text{ECHO}, id \rangle \) messages in Round 2, \( N - 2t \) of which must have come from the correct processes.

Proof of Lemma 4.3: By Lemma 4.2, all \( N - t \) correct ids are in \( timely \), therefore also in \( accepted \). It remains to calculate the maximum number of Byzantine ids that can be in \( accepted \). By Lemma A.1, each \( id \in accepted \) must have been echoed in Round 2 by at least \( N - 2t \) correct processes. This means that from the total of at most \( t(N-t) \) identifiers broadcast by the Byzantine processes in Round 1, \( \lfloor \frac{t(N-t)}{2} \rfloor = t + \lfloor \frac{t^2}{2t} \rfloor \) can be in \( accepted \) at the end of Round 4.

The following lemma is auxiliary and states that if we construct two multisets by adding pairwise values separated by some given distance from each other, then after we order the multisets, the entries on the corresponding indexes still preserve this distance.

Lemma A.2: Let \( U \) and \( W \) be two ordered multisets with \( k \) elements each, created by adding \( k \) pairs of elements \( a, pair(a) \) into \( U, W \) respectively, such that \( a + \delta \leq pair(a) \). Then, for any \( 1 \leq i \leq k \), \( u_i + \delta \leq w_i \).

Proof: We first show that the inequality holds for the first elements in the ordered multisets, i.e.

\[
u_1 + \delta \leq w_1.
\]

Since \( w_1 \) is the smallest in \( W \), \( w_1 \leq pair(u_1) \). If \( w_1 = pair(u_1) \), then (1) follows. If \( w_1 < pair(u_1) \), there exists \( u_1 \) such that \( w_1 = pair(u_1) \). Since \( u_1 \) is the smallest in \( U, u_1 + \delta \leq u_i + \delta \leq w_i \), as claimed.

Now, by making \( pair(u_1) \) a new pair of \( u_i \), the same argument is used to iteratively prove (1) for \( U = U \setminus \{u_1\} \) and \( W = W \setminus \{w_1\} \) until \( U \) and \( W \) are empty. Therefore, \( 1 \leq i \leq k, u_i + \delta \leq w_i \), as needed.

The following lemma shows that during the approximation procedure, the distance between the ranks of two ids included in the \( timely \) set of some correct process maintains at least \( \delta \).

Lemma A.3: If for some ids \( \langle id, id' \rangle \in timely \), at the beginning of Round \( r, ranks[id] + \delta \leq ranks[id'] \) and \( \lvert votes[id] \rvert, \lvert votes[id'] \rvert \geq N - t \), then at the end of Round \( r, ranks[id] + \delta \leq ranks[id'] \).

Proof: Since \( id, id' \in timely \), all votes accepted in Line 25 must contain new ranks for both \( id \) and \( id' \) spaced by at least \( \delta \). Hence, \( \lvert votes[id] \rvert = \lvert votes[id'] \rvert \).

If there are less than \( N \) entries in each set, the \( ranks[id] \) and \( ranks[id'] \) will be added respectively such that both sets have exactly \( N \) entries (Lines 10-11 of Alg. 3), (by assumption, the added values also preserve the distance of at least \( \delta \)).

Now, assume \( U, W \) are multisets resulted from ordering \( votes[id] \) and \( votes[id'] \) respectively. By Lemma A.2, for any \( 1 \leq i \leq N, u_i + \delta \leq w_i \). Hence, after deleting from \( U \) and \( W \), \( t \) smallest and \( t \) largest entries (Lines 13-14 of Alg. 3), it still holds that \( 1 \leq i \leq N - 2t, u_i + \delta \leq w_i \). The distance between the new values (calculated in Line 16) is given by

\[
\frac{\text{AVG}(select_t(W)) - \text{AVG}(select_t(U))}{\text{SUM}(select_t(U)) + t \delta} = \delta.
\]

Proof of Lemma 4.4: \( isValid(ranks_p, ranks_q) \) checks if the distance between the ranks of all elements in \( timely \) is at least \( \delta \). By Lemma 4.1, \( timely \subseteq accepted \). Therefore, if the entries in \( ranks_p \) preserve the distance of least \( \delta \), for any \( id \) such that \( id \in \bigcup_{q \notin q} \text{is correct timely}_q \in \text{Round } r \), then \( isValid(ranks_p, ranks_q) = true \).

We now show by induction on \( r \) that the distance between the ranks of ids in \( timely \) is preserved at least \( \delta \) by all correct processes in any Round \( r \geq 5 \). For the base case of \( r = 5 \), recall that \( p \) constructs the initial ranks in such a way that all ranks for the accepted set are spaced by at least \( \delta \) (Line 28 of Alg 1), therefore \( isValid(ranks_p, ranks_q) = true \).

For the induction round, assume that, for the rank held by \( p \) in Round \( r \), \( isValid(ranks_p, ranks_q) = true \). Therefore, for each element in \( timely \) each correct process will receive at least \( N - t \) valid votes. And since by assumption, the correct votes are valid in Round \( r \) and by Lemma 4.2 each correct vote contains new ranks for all ids in \( timely \), \( p \) will update their values in Line 35 of Alg 1 and, by Lemma A.3, the new ranks calculated by each correct process at the end of Round \( r \) preserve the necessary distance at least \( \delta \). Therefore, \( isValid(ranks_p, ranks_q) = true \) in \( r + 1 \).

Proof of Lemma 4.7: By assumption, \( id \in timely \), therefore, by Lemma 4.1, \( id \in accepted \). Also, by Lemma 4.2, all correct ids are in \( timely \) and \( timely \) and therefore in \( accepted \) at each correct process. Hence, \( \lvert accepted \setminus accepted \rvert \geq N - t \). On the other hand, by Lemma 4.3, all correct processes have \( \lvert accepted \rvert \leq N + t - 1 \).
Hence, the initial ranks calculated in Line 28 of Alg 1 of each common element of accepted_p and accepted_q differs by at most \((2t - 1) \times \delta\).

**Proof of Lemma 4.8:** Since \(id \in \text{timely}_p\), then by Lemma 4.4 and Corollary 4.5, \(\text{votes}_p[id]\) and \(\text{votes}_q[id]\) have at least \(N - t\) entries from the correct processes, therefore after executing Lines 12-14 of Alg. 3 both multisets have exactly \(N\) entries.

Let \(C\) be the multiset of ranks of \(id\) issued by all correct processes in Alg. 1, in Round \(r\). Note that \(C \subseteq \text{votes}_p[id], \text{votes}_q[id]\).

Let \(A, B\) be ordered multisets resulting from deleting \(t\) maximal values and \(t\) minimal values from \(\text{votes}_p[id]\) and \(\text{votes}_q[id]\), respectively. Let \(a_1 \leq \cdots \leq a_c\) be the elements of \(\text{SELECT}_t(A)\) and \(b_1 \leq \cdots \leq b_c\) be the elements of \(\text{SELECT}_t(B)\), where \(c\) is the number of elements selected. Note that \(c = \sigma_i\).

First, we need to show that, for \(1 \leq i \leq c - 1,\)

\[
\text{MAX}(a_i, b_i) \leq \text{MIN}(a_{i+1}, b_{i+1}). \tag{2}
\]

It suffices to show that \(a_i \leq b_{i+1}\), then by symmetric argument \(b_i \leq a_{i+1}\). Suppose, by contradiction, that \(a_i > b_{i+1}\). There are at least \(t(i+1)+1\) elements in \(B\) less than or equal to \(b_{i+1}\). By our supposition, these elements are strictly less than \(a_i\). However, there are at most \(ti\) elements in \(A\) strictly less than \(a_i\). Therefore, at least \(t(i+1)+1-2ti = t+1\) elements in \(B\), are not in \(A\). However, since \(|\text{votes}_p[id] \cap \text{votes}_q[id]| \geq N - t\), it holds that \(|A \cap B| \geq N - t - 2t\). Therefore,

\[
|B - A| = |B - (A \cap B)| \leq (N - 2t) - (N - 3t) = t.
\]

Hence the contradiction and (2) follows.

We then use (2) to prove the lemma. The discrepancy between \(\text{ranks}_p[id]\) and \(\text{ranks}_q[id]\), which are updated in Line 16 of Alg. 3 at the end of Round \(r\), is given by,

\[
\frac{1}{c} \sum_{i=1}^{c} \left(\text{MAX}(a_i, b_i) - \text{MIN}(a_i, b_i)\right)
\]

Expanding the sum and successively applying (2),

\[
\frac{1}{c} \sum_{i=1}^{c} (\text{MAX}(a_i, b_i) - \text{MIN}(a_i, b_i))
\]

\[
= \frac{1}{c} (\text{MAX}(a_c, b_c) - \text{MIN}(a_c, b_c))
+ \frac{1}{c} \sum_{i=1}^{c-1} (\text{MAX}(a_i, b_i) - \text{MIN}(a_i, b_i))
\]

\[
\leq \frac{1}{c} (\text{MAX}(a_c, b_c) - \text{MIN}(a_1, b_1)) \tag{4}
\]

On the other hand, since we deleted \(t\) extremal values from \(\text{votes}_p[id]\) and \(\text{votes}_q[id]\), it is true that \(\text{MAX}(a_c, b_c) \leq \text{MAX}(C)\), and \(\text{MIN}(a_1, b_1) \geq \text{MIN}(C)\). Therefore, the averages are within the interval of the input values belonging to the correct processes.

Moreover, from (3) and (4),

\[
|\text{AVG}(\text{SELECT}_t(A)) - \text{AVG}(\text{SELECT}_t(B))| \\
\leq \frac{1}{c} (\text{MAX}(C) - \text{MIN}(C)) \\
= \frac{1}{\sigma_i} \Delta_y.
\]

Hence, the lemma follows.

**Proof Lemma 4.9:** By successive applications of Lemma 4.8,

\[
\Delta_{r+1} \leq \left(\frac{1}{\sigma_i}\right)^r \Delta_1 \leq \left(\frac{1}{2}\right)^{\left[2\log(t)\right]} + 5 \times 2t \left(1 + \frac{1}{N + t}\right) \leq \frac{1}{6} (N + t).
\]

**APPENDIX B**

**Proof of Lemma 5.1:** By Lemma 4.3, the number of ids in the accepted set of any correct process is at most \(N + [\frac{t^2}{N-2t}] = N\). Due to the stretching factor of \(\delta = 1 + \frac{1}{N(t+3)}\), the initial ranks are bounded by \(N \times \delta\). Since by Lemma 4.8 the values returned by the approximation belong to the interval of the initial correct values, the rounded outputs will be at most \(\text{ROUND}(N \times \delta) = N\).

**Proof of Lemma 5.2:** By Lemma 4.7, the maximum discrepancy between the votes is at most \(t + [\frac{t^2}{N-2t}]\times \delta = t \times \delta\). On the other hand, by Lemma 4.8, the convergence rate of each approximation round is at least \(\sigma_i = [\frac{N-2t}{3t^3}] + 1 > [\frac{t^2}{2t}] + 1 = t + 1\). Therefore, after 4 convergence rounds, the values of the correct processes are within

\[
\frac{t \times \delta}{(t+1)^4} < \frac{1}{3t^3} \leq \frac{-1}{2}.\]
APPENDIX C

Proof of Lemma 6.1: For each echo message received in Round 2, a correct process compares the number of ids in common, that should be at least $N - t$ out of $N$ allowed per message (procedure ISVALID). Due to this sanity check, each Byzantine process can introduce only $2t$ Byzantine ids in an echo message: in the worst case, the Byzantine process includes $t$ Byzantine ids already known to the receiver and additional $t$ Byzantine ids. Therefore, the total number of echoes of Byzantine ids received from the Byzantine processes by each correct process in Round 2, is at most $2t^2$.

Proof of Lemma 6.2: Assume, by contradiction, $newid_p[id] - newid_p[id'] < N - t$. This is only possible if $counter_p[id] < N - t$ (Line 20). This means that, in Round 2, $p$ received less than $N - t$ echoes of $id$. It can only happen if some correct $p'$ did not echo $id$. This, in turn, is only possible if $p'$ did not receive $id$ in Round 1. But since $id$ is correct, it was sent to all the processes in Round 2. Contradiction.