ABSTRACT
In this paper we consider the fundamental problems of renaming and order-preserving renaming [1] in a synchronous message passing system with Byzantine failures. We study the feasibility of solving these problems using randomized algorithms under both non-rushing and rushing adversaries. We first show that there is a randomized algorithm that solves renaming efficiently for any \( t < N \) under the non-rushing adversary (\( N \) is the number of processes, and \( t \) is the maximum number of Byzantine processes). This result establishes a separation between randomized and deterministic renaming, since it is known that there are no efficient deterministic algorithms for \( t \geq N/3 \). Our algorithm terminates in \( O(\log N) \) rounds w.h.p. We next consider the renaming problem in the harder setting with the rushing adversary. Interestingly, we show that in this setting the algorithm also works with \( t = 1 \) but fails for larger \( t \). We then give an algorithm that works with any \( t < N \) by relying on cryptographic commitment. Finally, we turn our attention to the problem of order-preserving renaming, which requires the new names to preserve the order of the initial identifiers. For this problem, we prove a tight \( t < N/3 \) bound that holds for both deterministic and randomized algorithms.

Categories and Subject Descriptors
F.2.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity—Nonnumerical Algorithms and Problems

General Terms
Theory, Algorithms, Reliability

Keywords
Renaming problem; Byzantine failures; randomized algorithms; synchronous message passing model.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

PODC’13, June 22–24, 2013, Montréal, Québec, Canada.
Copyright 2013 ACM 978-1-4503-2065-8/13/07 ...$15.00.

1. INTRODUCTION
This paper concerns the fundamental problems of renaming and order-preserving renaming defined by Attiya, Bar-Noy, Dolev, Peleg, and Reischuk [1]. The renaming problem is informally described as follows: a set of \( N \) processes \( \{p_1, \ldots, p_N\} \) with unique ids from a possibly unbounded namespace must pick new names from a smaller bounded range \( [1, \ldots, M] \). The range of values to which new names belong is called the target namespace. If the size of the target namespace is equal to the number of processes, i.e. \( M = N \), the renaming is called tight. The order-preserving variant of the renaming problem requires the new names to preserve the ordering of the original ids (i.e., if \( p_i \) and \( p_j \) are correct processes and \( p_i \) has a smaller id than \( p_j \), then the new name of \( p_i \) must be smaller than the new name of \( p_j \)).

In this paper we consider a synchronous message-passing model where up to \( t \) processes may be Byzantine. In this model, renaming can be solved easily for any \( t < N \) with very inefficient algorithms whose round complexity is proportional to the highest correct process id [21]. We are interested in efficient algorithms, that is, algorithms whose round complexity is bounded by a function of \( N \) (not by the highest id). It has been shown that no such algorithms exist if \( t \geq N/3 \) [21], but this impossibility concerns only deterministic algorithms.

We show that, with the help of randomization, renaming can be solved efficiently for any \( t < N \). Therefore, our result implies that randomization is an important and necessary technique for solving renaming efficiently. To our knowledge, this is the first paper to study randomized solutions for renaming in this model.

When considering randomized algorithms, the literature considers two adversaries with different powers. With the non-rushing adversary, processes are forced to take steps simultaneously at the beginning of each round. Therefore, Byzantine processes must choose messages to send in each round before learning the random choices made by the correct processes in the same round. With the rushing adversary, Byzantine processes may execute each step after learning about the random choices of the correct processes. In [15], non-rushing and rushing adversaries are called simultaneous and sequential, respectively. In this paper, we study renaming under both adversaries.

We first note that it is possible to solve renaming using a consensus algorithm to agree on a set of identifiers. It is also known that sometimes randomization can be used to solve consensus [3] when deterministic algorithms fail [12]. Thus, one might wonder whether we could obtain an efficient
randomized renaming algorithm that tolerates $t < N$ Byzantine failures, as follows: first, obtain an efficient randomized consensus algorithm that tolerates $t < N$ Byzantine failures; then use this algorithm to solve renaming. Unfortunately, this approach would not work: it has been shown that with Byzantine failures and $t \geq N/3$, any randomized consensus algorithm fails with probability at least $1/3$ [15, 17]. This result holds even with the non-rushing adversary.

In seeking efficient algorithms for renaming, we first consider the weaker non-rushing adversary. We propose an algorithm based on a simple idea: each process randomly chooses a new id from the range $\{1, \ldots, N\}$ and applies tie-breaking rules to solve collisions. We show that this algorithm terminates in $O(\log N)$ rounds w.h.p.

We next consider the more challenging rushing adversary. We show that our first algorithm can tolerate $t = 1$ Byzantine failure but the algorithm fails if $t > 1$. This is because, in each round, Byzantine processes can first observe the choice of a correct process and mimic it, causing infinitely many collisions. To tackle this problem, we use a cryptographic commitment primitive to force processes to commit to a choice without revealing their value; this technique prevents Byzantine processes from constantly mimicking the choices of correct processes. Using cryptographic commitment, we show how to extend the previous algorithm to work for any $t < N$.

To use cryptographic commitment, we must assume a polynomially bounded adversary, so that the adversary cannot break the cryptographic primitive. We believe our use of cryptography is sensible in the following two ways. First, we do not assume a primitive that needs a public key infrastructure (PKI). By contrast, primitives such as digital signatures, which are frequently used to cope with Byzantine processes, require a PKI. The problem is that a PKI implies that processes have the public key of all processes, but the public key serves as a unique process id; and if processes have knowledge of unique ids for each other, the renaming problem becomes trivial (e.g., each process sorts these ids, and then outputs as its new name the position of its id in this sorted order).

Second, even if the adversary breaks the cryptographic primitive by luck (by guessing the random seed used by a correct process to commit a value), the algorithm never violates the properties of the renaming problem, though termination may get delayed. Even in this case, termination is ensured with probability one. By contrast, certain algorithms that rely on cryptography will fail with non-zero probability when the adversary is lucky [9].

We then turn our attention to the order-preserving renaming problem [1]. This variant is interesting in settings where the original identifiers encode some additional information, such as their relative priority in accessing a shared resource. We prove that it is impossible to solve order-preserving renaming if $t \geq N/3$, even if one could use randomized algorithms. Thus, randomization does not help solving this problem, in contrast to the renaming problem. To our knowledge, this is the first result that separates the resiliency of renaming and order-preserving renaming algorithms. Interestingly, the impossibility applies to a target namespace of any size. The proof reduces the case of $N > 3$ to the case of $N = 3$. It then considers a candidate algorithm for the case $N = 3$ and shows that it must fail, by constructing an indistinguishability ring of executions that violates the properties of order-preserving renaming. This technique was applied previously to the problem of consensus with Byzantine failures [11]. Here we extend it to order-preserving renaming, which is a weaker problem and hence harder to prove impossibility results for. In our proof, we construct a ring of size larger than the target namespace and use the following argument to establish a contradiction: roughly, we show that names must increase as we traverse the ring in one direction, but this exhausts the target namespace after going around the ring.

1.1 Summary of Contributions

Table 1 graphically presents our contributions, shown in gray, in the context of the known results.

In summary, we make the following contributions:

- We show that by using randomization, renaming can be solved in $O(\log N)$ rounds w.h.p. for any $t < N$ under a non-rushing adversary.
- We show that renaming can be solved in $O(\log N)$ rounds w.h.p. with $t = 1$, even under the rushing adversary.
- We show that renaming can be solved in $O(\log N)$ rounds w.h.p. for any $t < N$ under a polynomially bounded rushing adversary, by using cryptographic commitment.
- All our algorithms solve tight renaming.
- We show that order-preserving renaming cannot be solved if $t \geq N/3$; this impossibility applies to both deterministic and randomized algorithms.

1.2 Paper Organization

The remainder of this paper is organized as follows. In Section 2, we formalize the system model and in Section 3 we state the problem definition. Section 4 is dedicated to related work. In Section 5 we present a randomized renaming algorithm that works under the non-rushing adversary. In Section 6, we first show that our algorithm tolerates one Byzantine process even under the rushing adversary, and then we extend the algorithm to work for any $t < N$. In Section 7, we prove the $t < N/3$ bound for the order-preserving variant in both deterministic and randomized settings. Finally, in Section 8, we present conclusions and outline directions for future work.

2. SYSTEM MODEL

We consider a synchronous round-based message-passing system of $N \geq 3$ processes $p_1, \ldots, p_N$, where $N$ is known a priori. Each correct process has a unique identifier, originally known only to the process itself. Execution proceeds in rounds; in each round, a process can send messages to its neighbors, receive at most one message from each neighbor, and change state. In randomized algorithms, the process obtains a fixed-length string of private random bits before changing state. We consider a fully-connected network with
bidirectional links between every pair of processes. The bidirectional links of each process are numbered 1, \ldots, N; there is no global assignment of processes to link numbers, and different processes may have inconsistent link numberings. We denote by \( \text{link}(p_i) \) the number at \( p_i \) of the bidirectional link to \( p_j \). At each process, we assume that link number \( N \) is the self-loop, and that a process knows the number of the link from which it receives a message.

The system is subject to Byzantine failures of processes, as we explain next.

### 2.1 Power of the Adversary

Byzantine processes exhibit arbitrary behavior and are allowed to collude among themselves. We model the behavior of the Byzantine processes by an adversary that knows the protocol, can decide which processes to corrupt at any time during the execution, and has full control of the corrupted processes. We assume that the adversary does not have access to the private random bits of the correct processes, and we allow the adversary to corrupt at most \( t \) processes.

Additionally, we distinguish between rushing and non-rushing adversaries. The non-rushing adversary cannot decide the messages to be sent during a particular round based on the messages the corrupted processes receive during that same round. Under the rushing adversary, it is pessimistically assumed that the messages addressed to the Byzantine processes are always delivered immediately, and that the adversary has time to inspect the messages addressed to the corrupted processes before issuing its messages in the same round.

The adversary is allowed to eavesdrop the communication between the correct processes. However, the adversary is unable to send messages in the name of correct processes, or modify messages that the correct processes send.

### 2.2 Cryptographic Techniques

When we use cryptography in Section 6, we consider a computationally bounded adversary that is limited to computing polynomially bounded functions. We assume the availability of a cryptographic commitment scheme. Roughly speaking, such a scheme has two separate stages, commitment stage and revealing stage, with the following interface for each respective stage:

- **\text{commit}(value)**: a process generates a cryptographic commitment to a value, without revealing the value.
- **\text{reveal}(value)**: a process reveals the value, which must match the previously announced commitment.

The properties of a commitment scheme can be informally stated as follows.

- **Hiding**: it is computationally hard for the receiver to know in the commitment stage the value to which the sender commits.
- **Binding**: it is computationally hard for the sender to commit more than one value and to reveal a value that it has not committed.

For a more formal treatment, we refer the reader to [14].

### 3. PROBLEM DEFINITION

The renaming problem can be precisely defined as follows. Each process has an initial id in some original namespace, and it must decide on a new name in the same namespace \( \{1, \ldots, M\} \), where \( M \geq N \) is a parameter of the problem, such that the following conditions must hold [1]:

- **Termination**: Each correct process eventually decides on a new name.
- **Validity**: If a correct process decides, it decides on a new name in \( \{1, \ldots, M\} \).
- **Uniqueness**: No two correct processes decide on the same new name.

For randomized algorithms, the termination property is weakened to

- **Termination with probability 1**: With probability 1, every correct process eventually decides on a new name.

When \( M = N \) the problem is called tight renaming. The stronger order-preserving variant of renaming is obtained by adding the following property:

- **Order-Preserving [1]**: New names of the correct processes preserve the order of the initial ids.

In general, algorithms can have unbounded running time. In this work we are interested in efficient algorithms, whose round complexity can be bounded by a function of \( N \), the number of processes (this property is also known in the literature as strong termination [8]). For randomized algorithms, we require the probabilistic round complexity of an efficient algorithm to be bounded by a function of \( N \).

### 4. RELATED WORK

Related work concerns other solutions and lower bounds for the renaming problem (Section 4.1), techniques from cryptography (Section 4.2), and the symmetry breaking problem (Section 4.3).
4.1 Renaming Problem

The renaming problem was originally introduced in [1] for the asynchronous message-passing model with crash failures. The authors presented renaming and order-preserving renaming algorithms for $t < N/2$. This bound was also shown to be optimal.

The first paper to address the renaming problem in synchronous message-passing systems prone to Byzantine failures is [21], which shows that renaming can be solved deterministically for any $t < N$ with the following algorithm: a process waits until the round corresponding to the value of its id, then picks an available name, and sends its decision to other processes. The remaining processes exclude the announced decision from the available names. Since all correct processes have different ids, no two correct processes decide in the same round, and hence they always choose distinct names. This algorithm, however, is not efficient because its running time depends on the values of original ids, which can be taken from an unbounded namespace. The same paper shows that the unbounded running time is required for deterministic renaming if $t \geq N/3$. It is also shown that a weaker renaming problem with a bounded original namespace can be solved in $O(N \log[N_{\text{max}}/N])$ rounds for $t < N$, where $N_{\text{max}}$ is the size of the original namespace. The paper also presents a renaming algorithm for $t < N/3$ with round complexity of $O(\log N)$. The algorithm is based on the crash-tolerant renaming algorithm introduced in [5] and the automatic crash-to-Byzantine translation techniques introduced in [2, 20] with modifications to avoid using process ids.

The order-preserving renaming problem in the same model was studied in [7], which proposes an algorithm for $t < N/3$ with $O(\log N)$ round complexity. This algorithm is based on Byzantine-tolerant approximate agreement and works as follows. Processes exchange their ids, order the set of the received ids, and propose for each id a new name, which is the rank of that id in the ordered set. Then, processes run a coordinated Byzantine-tolerant approximate agreement on the rank of each id.

4.2 Cryptography

As mentioned earlier, the usual authentication-based approach to mitigate the power of the Byzantine processes cannot be applied in renaming, because the unique identities of processes are not known a priori. We circumvent this difficulty by using a cryptographic commitment primitive. The concept of commitment was first formally introduced in [19] but appeared earlier in [4, 10] and is central in cryptographic protocol design. Often described as the digital analogue of sealed envelopes, commitment schemes enable a party, known as the sender, to commit itself to a value (binding property) while keeping it secret from the receiver (hiding property).

Since it is impossible to have both binding and hiding properties against an unbounded adversary [14], the existing commitment protocols usually provide the following properties. In the statistically-binding commitment, e.g. [22], the binding property holds with overwhelming probability against the computationally bounded adversary. On the other hand, in the statistically-hiding commitment, e.g. [6], the hiding property holds with overwhelming probability against the computationally bounded adversary.

4.3 Randomized Symmetry Breaking

Finally, the randomization techniques used in this paper bear some similarities to the techniques employed in the symmetry breaking problem. A notable example is the work by Itai and Rodeh [16] that addresses symmetry breaking in anonymous rings. The paper presents a randomized ring leader election algorithm that works as follows. Each process randomly picks a value from a given range and sends it through the ring. The process that has picked the largest value becomes a leader. In case of a collision, the processes with the largest value repeat the algorithm until one process wins. Several other papers applied randomization techniques to the leader election problem, e.g. [13, 18]. To our knowledge, Byzantine failures were not considered in this line of research.

5. Byzantine Renaming Under the Non-Rushing Adversary

In this section we consider the non-rushing adversary and present an algorithm (Alg. 1) that solves renaming efficiently. In fact, the algorithm solves tight renaming, where $M = N$.

The algorithm is based on the following idea. A process chooses uniformly at random a name from $\{1, \ldots, N\}$ and exchanges its choice with all other processes. If no other process picks the same name, the process is done and informs all other processes. Otherwise, the process excludes names already chosen by other processes and then restarts. Since the adversary cannot inspect the random choices of the correct processes before issuing its messages, each process will eventually pick a unique name. All Byzantine processes can do is to repeatedly claim $t$ distinct available names to increase the chance of collisions. For example, in the presence of $N - 1$ Byzantine processes, the correct process has a chance of $1/N$ of not colliding with the choices of the Byzantine processes. Hence, without additional rules, the expected decision time for a correct process could be linear in the size of the network.

To speed up the decision time, we use a tie breaking rule that allows processes to decide even in the case of collisions. Tie breaking is done by appointing in each phase a set of processes whose choices, in case of a collision, have priority over the choice of the given process. These sets are calculated in each phase as follows. In odd phases, the set consists of the undecided neighbors with ids lower than myId (the id of the current process); in even phases, the set consists of undecided neighbors with ids higher than myId. Roughly speaking, with this rule, in a given phase, a correct process competes with some fraction of undecided processes, and, in the following phase, competes with the remaining fraction of undecided processes. As we will show, this tie breaking rule reduces the decision time to a factor of $\log N$. Hence, the algorithm terminates in $O(\log N)$ rounds w.h.p (i.e., with probability at least $1 - \frac{1}{N}$), where $c > 0$ is a constant.

We now describe the algorithm in detail. Each process $p_i$ stores the following data structures:

- $\text{id}_i$: an array that stores ids (not names) that $p_i$ knows about, indexed by the link to which they are connected, i.e. $\text{id}_i[j]$ stores the id of the neighbor connected to link $j$ of $p_i$. Initially, $\text{id}_i[j] = \perp$ for every $j$.
- $\text{freenames}_i$: the set of names that $p_i$ believes to be free
Algorithm 1 Renaming with $t < N$

01 Initialization:
02 undecided := \{1, \ldots, N\};  // set of links to undecided neighbors
03 freenames := \{1, \ldots, N\};  // set of available names
04 foreach $i \in \{1, \ldots, N\}$ do
05 \hspace{1em} ids[i] := 1;  // array that stores old ids of all neighbors
06
07 In Phase 1 do
08 \hspace{1em} // Phase 1 has a single round
09 send (ID, myId) to all links;
10 foreach $i \in \{1, \ldots, N\}$ do
11 \hspace{2em} if (ID, id) has been received from link $i$ then
12 \hspace{3em} ids[i] := id;
13 \hspace{2em} proceed to Phase 2;
14
15 In Phase $\phi > 1$ do
16 \hspace{2em} // choose priority set for the current phase
17 \hspace{3em} if $\phi$ is odd then
18 \hspace{4em} plinks := \{ $i \in \text{undecided} : \text{ids}[i] < \text{myId}$\};
19 \hspace{3em} else
20 \hspace{4em} plinks := \{ $i \in \text{undecided} : \text{ids}[i] > \text{myId}$\};
21 \hspace{2em} myName := select an element in freenames uniformly at random;
22 \hspace{2em} send (PROPOSAL, myName) to every link $i$ such that $i \in \text{undecided} \setminus \text{plinks}$;
23 \hspace{2em} \hspace{1em} // check for collisions
24 \hspace{3em} if $\exists i \in \text{plinks} : (\text{PROPOSAL}, \text{name}_i)$ has been received from link $i$ such that myName = name, then
25 \hspace{4em} winner := false;
26 \hspace{3em} else
27 \hspace{4em} winner := true;
28 \hspace{2em} \hspace{1em} // Round 2
29 \hspace{3em} if winner = true then
30 \hspace{4em} send (DECIDED, myName) to every link $i$ such that $i \in \text{undecided};$
31 \hspace{4em} return myName;
32 \hspace{3em} else
33 \hspace{4em} foreach $i \in \text{undecided}$ do
34 \hspace{5em} if (DECIDED, name$_i$) has been received from link $i$ then
35 \hspace{6em} undecided := undecided \{ $i$\};
36 \hspace{5em} freenames := freenames \{ name$_i$\};
37 \hspace{4em} proceed to Phase $\phi + 1$;

(not selected by its neighbors). Initially, freenames = \{1, \ldots, N\} (all possible names).

- undecided: the set of links to neighbors that $p_i$ believes to not have decided yet. Initially, undecided = \{1, \ldots, N\} (all links).

The purpose of the first phase of the algorithm is for processes to exchange their initial ids. Each process sends its own id to all links and stores the ids it receives from other links (Lines 07-10). The ids received in phase 1 are used in all subsequent phases.

The following phases are tournament phases, where processes compete for available names. There are two ways of winning a tournament: if there are no collisions on the name chosen by a process in the current phase (i.e., the process was the only one to make that choice) or, when collisions occur, by winning a tie breaking rule, which differs according to the phase number. The tie breaking rule updates plinks in each phase as follows. In odd phases, plinks includes undecided neighbors with ids lower than \text{myId}; in even phases, plinks includes undecided neighbors with ids higher than \text{myId} (Lines 13-16). In what follows, recall that link$_{\phi}(p_i)$ denotes the number at process $p_i$ of the bidirectional link to $p_i$ (see Section 2). The relevant property of the plinks assignment is that, for any phase $\phi$, when considering any two correct processes $p_i$ and $p_j$, if link$_{\phi}(p_i) \in \text{plinks}_i$, then link$_{\phi}(p_j) \notin \text{plinks}_j$. Furthermore, if link$_{\phi}(p_i) \in \text{plinks}_i$, in $\phi$, and $p_i$ and $p_j$ do not decide in $\phi$, then link$_{\phi}(p_i) \in \text{plinks}_i$ in $\phi + 1$.

Each tournament phase has two rounds. In the first round, a correct process selects plinks, picks randomly a name from freenames, and sends the name to the undecided neighbors that do not belong to plinks (Lines 17-18). There is no need to send the name to the neighbors in plinks because if a collision occurs in this phase, they will ignore the choice of the given process. A process wins the tournament if it has not received the same name from any process in plinks. At the end of the first round, processes check if they have won the tournament (Lines 19-22).

In the second round, if a correct process has won the tournament, it sends a (DECIDED, myName) message to all undecided neighbors and terminates, returning variable myName as its new name (Lines 23-25). Otherwise, if the process did not win, it collects (DECIDED, name$_i$) messages from its neighbors, removes these neighbors from undecided (doing so excludes these neighbors from all succeeding tournaments), and removes the names elected by the decided processes from freenames (Lines 26-30).

Each undecided process keeps executing consecutive tournament phases until it wins a name in one of the tournaments.

Analysis

In the following we prove the correctness of Alg. 1. Whenever needed to distinguish between local variables at distinct processes, we use subscript $i$, to indicate the local variables of process $p_i$. Superscript $\phi$ indicates the value of a variable in Round 1 of phase $\phi$. 
Lemma 1. For any phase $\phi > 0$ and any correct processes $p_i$ and $p_j$, if $p_i$ has not decided a new name before $\phi$, then $\text{link}_j(p_i) \in \text{undecided}$, in Round $1$ of $\phi$.

Proof. Assume there exist correct undecided processes $p_i$ and $p_j$ such that $\text{link}_j(p_i) \notin \text{undecided}$, in Round $1$ of $\phi$. By the algorithm, the links to all neighbors are initially in $\text{undecided}$. (Line 02) and are excluded from the set only when a \{\text{DECIDED,}\} message is received from the corresponding neighbor (Lines 28-29). Thus, $p_i$ must have previously received \{\text{DECIDED,} $v_j$\} from $p_j$. This, in turn, means that $p_i$, sent \{\text{DECIDED,} $v_j$\} before $\phi$ (Lines 24-25). But by assumption, $p_i$ is undecided in Round $1$ of $\phi$—a contradiction.

The following lemma states that each correct process considers at least as many names as there are processes participating in each tournament phase.

Lemma 2. For every correct process $p_i$, in Round $1$ of any phase $\phi \geq 2$, $|\text{undecided}| \leq |\text{freenames}|$.

Proof. By induction on the phase number.

Base case. By the algorithm, $p_i$ starts with $|\text{undecided}| = |\text{freenames}| = N$ (Lines 02-03). Thus, in Round $1$ of phase $\phi = 2$, $|\text{undecided}| = |\text{freenames}|$.

Induction step. Assume $|\text{undecided}| \leq |\text{freenames}|$ in Round $1$ of phase $\phi$. In Round $2$ of the same phase, $p_i$ only accepts \{\text{DECIDED,} $v_j$\} messages from the links in $\text{undecided}$, (Line 27-28). For each link $j \notin \text{undecided}$, if $\text{link}_j(p_i)$ has been received from $j$ in the current phase, $p_i$ removes $j$ from $\text{undecided}$; and, if $v_j \in \text{freenames}$, $p_i$ also removes $v_j$ from $\text{freenames}$ (Lines 28-30). As a result, in Round $1$ of phase $\phi + 1$, $|\text{undecided}| \leq |\text{freenames}|$.

The following lemma establishes the uniqueness property of the algorithm based on two following observations: in case of a collision, at most one correct process (with the smallest id in odd phases and with the largest id in even phases) wins the tie breaking; the winning process always announces its decision before returning.

Lemma 3. No two correct processes decide on the same name in Alg 1.

Proof. Assume, by contradiction, that there are two correct processes $p_i$ and $p_j$ that decide on the same name $v$ in phases $\phi_i$ and $\phi_j$ respectively. We will distinguish two possible scenarios.

Case 1. $\phi_i \neq \phi_j$. Without loss of generality, assume $\phi_i < \phi_j$. By the algorithm, if $p_i$ decided on $v$ in $\phi_i$ then $p_i$, sent \{\text{DECIDED,} $v$\} in Round $2$ of $\phi_i$ to all undecided neighbors before returning the new name (Lines 24-25). By Lemma 1, $\text{link}_i(p_i) \notin \text{undecided}$, and $\text{link}_i(p_j) \notin \text{undecided}$, in Round $1$ of $\phi_i$. Therefore, $p_i$ sent $v$ to $p_j$, and $p_j$ excluded $v$ from $\text{freenames}$ in Round $2$ of $\phi_i$ (Lines 27-30). On the other hand, if $p_j$ decided on $v$ in phase $\phi_j$, then $p_j$ must have chosen $v$ from $\text{freenames}$ in $\phi_j$ (Line 17), which contradicts the previous statement.

Case 2. $\phi_i = \phi_j = \phi$. Without loss of generality, assume that $id_i < id_j$ and $\phi$ is odd (for even values of $\phi$ the argument is symmetric). If both $p_i$ and $p_j$ decided on $v$ in Round $2$ of $\phi$, then $p_i$ and $p_j$ were both undecided in Round $1$ of $\phi$.

Also, by Lemma 1, $\text{link}_i(p_i) \notin \text{undecided}$, and $\text{link}_j(p_j) \notin \text{undecided}$, in Round $1$ of $\phi$. Processes $p_i$ and $p_j$ randomly picked $v$ from $\text{freenames}$ and sent $v$ to their neighbors in $\text{undecided}$ \{\text{plinks}\} (Lines 17-18). By assumption, $id_i < id_j$; therefore, $\text{link}_j(p_i) \notin \text{undecided}$ \{\text{plinks}\}. Hence, $p_i$, sent $v$ to $p_j$ in Round $2$ (Line 18). Since $\text{link}_j(p_i) \in \text{plinks}$, $p_j$ received $v$ from $p_i$ (Lines 19-20). But by assumption, $p_j$ decided on $v$ in Round $2$, which is possible only if $p_j$ had not received $v$ from any link in $\text{plinks}$, in Round $1$ (Lines 19-22)—a contradiction.

Complexity

In the following we calculate the round complexity of the algorithm. Recall that we consider the non-rushing adversary, whose behavior is independent from the random choices of the correct processes in the current round.

We now prove that each correct process decides w.h.p. after $O(\log N)$ rounds. To do so, we consider an arbitrary correct process $p_0$ and give an upper bound on the probability that $p_0$ has not decided after $O(\log N)$ rounds.

Intuitively, the adversary can decrease the probability that $p_0$ decides in some phase by making certain Byzantine processes (those not in $\text{plinks}$) announce their decision in the previous phase. However, once a process decides, it is excluded from subsequent tournaments, and so the adversary can do this only a limited number of times. We will show that the adversary has to make a large fraction of processes to decide in order to decrease by a constant factor the probability that $p_0$ does not decide. Thus, in each phase, the adversary has to carefully balance the number of processes that it causes to decide and the probability that $p_0$ does not decide. We will show that, whatever the strategy of the adversary, after $O(\log N)$ rounds, $p_0$ decides w.h.p.

Lemma 4. Under the non-rushing adversary, the probability that a correct process $p_0$ decides by round $12 \log N + 3$ is at least $1 - \frac{1}{N}$.

Proof. In the analysis, we assume without loss of generality that the adversary controls all processes other than $p_0$; if the probability lower bound holds in this case, then it also holds if the adversary can control a smaller number of processes. We say that a link $\ell$ decides at an epoch $e$ (or at a round $r$) if $p_0$ receives a \{\text{DECIDED,} $v$\} message from link $\ell$ in epoch $e$ (or round $r$). Intuitively, we are considering a $p_0$-centric view of the system since $p_0$ is the only correct process. We say that link $\ell$ is undecided at epoch $e$ (or round $r$) if $\ell$ has not decided at epoch $e$ (round $r$) or earlier.

Recall that the algorithm is organized in phases of two rounds each. We will further group two consecutive phases into an epoch $e \geq 1$: epoch $e$ has phases $2e + 1$ and $2e + 2$. For convenience we also define epoch $0$ as having just a single phase, phase $2$.

In each epoch, we pick one phase to scrutinize more carefully; this is called the chosen phase of the epoch. For the other phase, we will use a trivial upper bound of $1$ on the probability of non-termination. The chosen phase of epoch $e$ is determined as follows. At the beginning of last round before epoch $e$, consider the sets $p[l] = \{i \in \text{undecided} : \text{id}_0[i] < \text{myId}_0\}$; and $p[l] = \{i \in \text{undecided} : \text{id}_0[i] > \text{myId}_0\}$, where $\text{undecided}_0$, $\text{id}_0[i]$, and $\text{myId}_0$ are variables of process $p_0$. If $|p[l]| < |p[l]|$ then we pick the first phase of epoch $e$ as the chosen phase, otherwise we pick the second phase.
phase. Intuitively, we want the phase with fewest priority links, where \( pl1 \) is the priority links of the first phase and \( pl2 \) is the priority links of the second phase (see Lines 13-16 of Alg.1). We let \( pl_e \) and \( npl_e \) be the priority links and non-priority links, respectively, of the chosen phase evaluated at the beginning of last round before epoch \( e \). That is, \( pl_e = pl1, npl_e = pl2 \) if the chosen phase is the first phase, and \( pl_e = pl2, npl_e = pl1 \) if the chosen phase is the second phase.

For each epoch \( e \), we define the following variables:

- \( N_e \): the number of undecided links at the beginning of the last round before epoch \( e \);
- \( \gamma_e = |pl_e|/N_e \);
- \( \alpha_e = \frac{|\text{links in } npl_e \text{ that decide before the chosen phase of } e|}{N_e} \);
- \( \bar{\alpha}_e = 1 - \alpha_e \);
- \( c_e \): number of links in \( pl_e \) that decide before the chosen phase of epoch \( e \).

We now bound the probability \( P_e \), that \( p_0 \) does not decide in the chosen phase of epoch \( e \), as follows:

\[
P_e \leq \frac{N_e \gamma_e - c_e}{N_e \bar{\alpha}_e - c_e} \leq \frac{N_e \gamma_e}{N_e \bar{\alpha}_e} \leq \frac{1}{2\bar{\alpha}_e}
\]  

(1)

Here, the first inequality holds because the best strategy for the adversary to delay termination is to claim as many free names as possible, and the second inequality is an upper bound of the total number of free names (at least \( N_e \bar{\alpha}_e - c_e \), the number of undecided links). The second inequality holds because \((a - c)/(b - c) < a/b\) for any positive integers \( a, b, c \) such that \( a < b \) and \( c < b \). The third inequality holds because \( \gamma_e \leq 1/2 \) by definition of \( \gamma_e \) and of the chosen phase.

We can trivially upper bound by 1 the probability that \( p_0 \) does not decide in the non-chosen phase of epoch \( e \). Thus, from (1), the probability that \( p_0 \) does not decide in epoch \( e \) is upper bounded by \( \frac{1}{2\bar{\alpha}_e} \) as well. Therefore, the probability \( P_e \) that \( p_0 \) does not decide in any of epochs \( 1, \ldots, e \) is upper bounded by

\[
P_e \leq P_1 \times \cdots \times P_e \leq \frac{1}{2^\sum_{1}^{e} \bar{\alpha}_e}
\]  

(2)

Note that \( N_{e+1} \leq N_e \bar{\alpha}_e \) because at least \( \alpha_e, N_e \) links decide before the chosen phase of epoch \( e + 1 \). Therefore,

\[
N_{e+1} \leq N_1 \bar{\alpha}_1 \bar{\alpha}_2 \cdots \bar{\alpha}_e
\]  

(3)

We do not know the values of \( \bar{\alpha}_e \) because they depend on the strategy of the adversary. However, from (2) and (3), we can upper bound the product \( P_e N_{e+1} \):

\[
P_e N_{e+1} \leq N_e / 2^e \leq N / 2^e
\]  

(4)

For \( e = 3 \log N \), we obtain \( P_e N_{e+1} \leq 1/N^2 \). If \( p_0 \) has not decided by the end of epoch \( e \) then \( 1 \leq N_{e+1} \). Therefore, \( P_e \leq P_e N_{e+1} \), and so \( P_e \leq 1/N^2 \). Note that each epoch has two phases, and each phase has two rounds. Therefore, there are three initial rounds before epoch 1. Thus, there are \( 3 + 12 \log N \) rounds until the end of epoch \( e \). Therefore, the probability that process \( p_0 \) has not decided by round \( 3 + 2 \log N \) is upper bounded by \( 1/N^2 \). \( \square \)

We now have all ingredients necessary to prove the correctness of Alg. 1.

**Theorem 5.** Under the non-rushing adversary, all correct processes decide on a new name in \( O(\log N) \) communication rounds w.h.p.

**Proof.** By Lemma 4, the probability that a correct process decides in at most \( 12 \log N + 3 \) rounds is at least \( 1 - 1/N \). By taking a union bound, the probability that there exists at least one correct process that has not decided after \( 12 \log N + 3 \) rounds is at most \( 1/N \). \( \square \)

We now have all ingredients necessary to prove the correctness of Alg. 1.

**Theorem 6 (Correctness).** Under the non-rushing adversary, Alg. 1 implements tight renaming for any \( t < N \).

**Proof.** Validity condition is satisfied by the algorithmic construction: processes propose new names from the set \( \{1, \ldots, N\} \) and decide only on the values they have proposed. Uniqueness follows from Lemma 3. Termination with probability 1 follows from Theorem 5. \( \square \)

### 6. Renaming Under the Rushing Adversary

In this section we consider the rushing adversary, which is allowed to inspect the messages from correct processes before having Byzantine processes send their own messages. As a result, in our algorithm, the Byzantine processes can echo the choices of each correct process causing infinitely many collisions. Surprisingly, even in this case, correct processes are still able to decide on new names in the presence of one Byzantine process.

#### 6.1 Case of \( t=1 \)

We now show that the algorithm of Section 5 works under the rushing adversary for \( t = 1 \). The validity property
follows from the algorithm construction. Uniqueness follows from Lemma 3.

It remains to show the termination with probability 1. Consider an execution of Alg. 1 with $t = 1$. For any $\phi > 2$, if a correct process has a link to the Byzantine process in the plinks set of phase $\phi$, then this process will ignore the choice of the Byzantine process in $\phi + 1$. As a result, the Byzantine process can influence the outcome of a half of tournament phases, while in the remaining phases the correct process is competing only with the undecided correct neighbors.

**Theorem 7.** Under the rushing adversary with $t = 1$, Alg. 1 terminates in $O(\log N)$ rounds w.h.p.

**Proof.** Let $p_i$ be an undecided correct process. For any two consecutive phases there exists a phase, say $\phi$, when the link to a Byzantine process $\notin \text{plinks}\phi$. Therefore, the choice of the Byzantine process is ignored by $p_i$ in every such $\phi$ (Line 19). Since by Lemma 2, in each phase there are as many free names as undecided processes, the expected decision time for $p_i$ is at most double the decision time under the non-rushing adversary (see Lemma 4 and Theorem 5).

### 6.2 General Case of $t > 1$

If $t > 1$, two Byzantine processes can announce to a correct process $p_i$ a smaller id and a larger id than $p_i$’s identifier, and then generate collisions deterministically in each tournament phase. In this way, the correct processes are never able to decide. To prevent such behavior, we introduce in Alg. 1 a cryptographic commitment primitive. In the modified algorithm, depicted in Alg. 2, the random choices of processes are not announced immediately. Instead, the undecided processes first commit to their choices without revealing the actual values, and in a subsequent round reveal their choices. Thus, the adversary is required to commit the values of the corrupted processes without knowing the values committed by the correct processes (hiding property). Furthermore, the adversary is not able to modify its choices during the revealing stage (binding property). Therefore, the algorithm operates analogously to its non-rushing counterpart.

As noted before in the text, with small probability, the adversary is able to break the commitment. In this case, it will be able to reproduce the value of a correct process, causing a collision. However, since in different phases processes use independent instances of commitment, correct processes still decide with probability 1. Therefore, under the computationally bounded adversary, with probability 1, Alg. 2 terminates correctly.

Commitment abstraction can be implemented in constant number of rounds, e.g. [22]. Hence, the total round complexity of Alg. 2 is logarithmic.

### 7. ORDER-PRESERVING RENAMING

In this section we study the order-preserving variant where the new names are required to preserve the order of the initial ids. We first establish the separation result for order-preserving renaming by showing that there is no deterministic algorithm with $N \leq 3t$ that implements order-preserving renaming. Our proof method is based on the indistinguishability argument widely used in the literature on Byzantine fault tolerance, e.g. [11, 17, 15].

We first give the impossibility proof for deterministic algorithms in the system with $N = 3$ and $t = 1$. In the proof, we consider the easiest case of order-preserving renaming from a bounded original namespace of size $M + 1$ into a target namespace of size $M$, for some arbitrary $M \geq N$. This implies a fortiori that the impossibility applies to namespaces of any size, as long as the original namespace is larger than the target namespace (otherwise the problem becomes trivial).

We take a candidate algorithm for the case $N = 3$ and construct an indistinguishability ring of executions, which violates the properties of order-preserving renaming. Namely, we construct a ring of size $M + 1$ and assign inputs in such a way that new names must increase as we traverse the ring in one direction, but this exhausts the target namespace after going around the ring.

**Theorem 8.** There is no deterministic algorithm that solves order-preserving renaming in a system with $N = 3$ and $t = 1$.

**Proof.** Assume by contradiction there exists an algorithm $\pi$ that solves order-preserving renaming for $N = 3$ and $t = 1$ from the original namespace of size $M + 1$ into the target namespace of size $M$, for some $M \geq 3$. Without loss of generality we assume $\pi$ is a full-information algorithm.

Consider a system with $M + 1$ processes $p_0, \ldots, p_M$. We consider an execution $\alpha$ of $\pi$ in this system, where processes are arranged in a ring as depicted in Fig. 1, and process $p_i$ has original id $i$. This is an execution with $M + 1 > N$ processes, even though algorithm $\pi$ is designed for $N = 3$.
processes, but we will argue this execution has some interesting properties. For every pair of adjacent processes in the ring, their view of the system is indistinguishable from the view in a 3-process system in which the two processes are connected to a corrupted third process. For instance, processes \( p_0 \) and \( p_1 \) cannot distinguish execution \( \alpha \) from execution \( \alpha_0 \) in Fig. 1 where both \( p_0 \) and \( p_1 \) are connected to a single corrupted process that simulates \( p_M \) and \( p_2 \). That is, the single Byzantine process sends to \( p_0 \) exactly what \( p_M \) sends to \( p_0 \) in \( \alpha \), and the same Byzantine process sends to \( p_1 \) exactly what \( p_2 \) sends to \( p_1 \) in \( \alpha \) (in Fig. 1, the gray area in execution \( \alpha_0 \) depicts the system simulated by a single Byzantine process). By assumption, in \( \alpha_0 \) processes \( p_0 \) and \( p_1 \) decide on valid names in the correct order. Moreover, since \( p_0 \) and \( p_1 \) cannot distinguish execution \( \alpha \) from execution \( \alpha_0 \), they decide in \( \alpha \) exactly on the same names as in \( \alpha_0 \). Similarly, \( p_1 \) and \( p_2 \) cannot distinguish \( \alpha \) from \( \alpha_1 \) in Fig. 1, and so on for each pair of adjacent processes in the ring. Therefore, in \( \alpha \), each pair of adjacent processes decides on valid names in the correct order.

By the validity property, in \( \alpha \) process \( p_0 \) decides on a new name \( v_0 \) such that \( 1 < v_0 < M \). By the order-preserving property, \( p_1 \) decides on name \( v_1 \) such that \( 1 < v_0 < v_1 \). Applying the order-preserving property to the new names of all pairs of processes from \( p_0 \) to \( p_M \), we see that process \( p_M \) decides on a new name \( v_M \) such that \( 1 < v_0 < v_1 < \ldots < v_{M-1} < v_M \). Thus, \( v_M > M \). But by the validity property, \( v_M \leq M \)—a contradiction to the existence of algorithm \( \pi \).

Theorem 8 is generalized to the case of \( t = \lceil N/3 \rceil \) by having three processes simulate the \( N \)-process system as described in [21]. More precisely, if there is an algorithm \( \pi \) for \( N \) processes and \( t = \lceil N/3 \rceil \), we can use \( \pi \) to obtain an algorithm for three processes and \( t = 1 \), which contradicts Theorem 8. The algorithm for three processes works as follows. Each process simulates \( \lceil N/3 \rceil \) or \( \lceil N/3 \rceil \) processes running algorithm \( \pi \), for a total of \( N \) simulated processes, where the simulated processes start with names that respect the id ordering of the three processes; each process then decides on the new name of any of the processes that it simulates.

Theorem 8 concerns deterministic algorithms. We now consider randomized algorithms. We use the indistinguishability argument to show that any randomized algorithm with \( N = 3 \) has a non-zero probability of error when running in the system depicted in Fig. 1. Again, we assume the easiest case of order-preserving renaming from the original namespace of size \( M + 1 \) into the target namespace of size \( M \), for some arbitrary \( M \geq N \). In our proof, we assume the non-rushing adversary, which is weaker than the rushing counterpart. Therefore, the impossibility holds under both adversaries.

Theorem 9. Under the non-rushing adversary, there is no randomized algorithm that solves order-preserving renaming in a system with \( N = 3 \) and \( t = 1 \).

Proof. Assume that there exists a (full information) randomized algorithm \( \pi' \) that solves order-preserving renaming for \( N = 3 \) and \( t = 1 \) from the original namespace of size \( M + 1 \) into the target namespace of size \( M \), for some \( M \geq 3 \).

Consider a system composed by \( M+1 \) processes \( p_0, \ldots, p_M \) arranged in a ring as depicted in Fig. 1. We show that there exists a finite execution in the ring such that all \( M + 1 \) processes terminate. From that point, the proof proceeds as in Theorem 8. The proof of the existence of such execution is slightly technical but follows from the termination with probability 1 of \( \pi' \). The proof will construct increasingly larger executions in which, successively, processes \( p_0, \ldots, p_M \) terminate, by arguing for each process \( p_i \) that it would terminate with probability 1 in a 3-process system with \( \pi' \). More precisely, for each pair of processes \( p_i \) and \( p_{i+1} \) the execution in the ring is indistinguishable from an execution \( \alpha_i \) where \( p_i \) and \( p_{i+1} \) is connected to a single Byzantine process that behaves exactly like \( p_{i-1} \) and \( p_{i+2} \) in \( \alpha \) (the arithmetic of indices is done modulo \( M + 1 \)). From the termination with probability 1 property of order-preserving renaming, \( \pi' \) all correct processes in a 3-process system running \( \pi' \) with \( t = 1 \) terminate with probability 1. We now consider executions of \( \pi' \) in the ring of \( M + 1 \) processes. We first claim that there is a finite execution \( \beta_0 \) of \( \pi' \) in the ring such that \( p_0 \) and \( p_1 \) terminate. This follows from (*) and the fact that an execution in the ring is indistinguishable by \( p_0 \) and \( p_1 \) from an execution in a 3-process system. We now claim that we can extend execution \( \beta_0 \) such that \( p_2 \) also terminates. Indeed, \( \beta_0 \) is indistinguishable by \( p_1 \) and \( p_2 \) from an execution in a 3-process system. Since \( \beta_0 \) is finite, it occurs with positive probability \( p_0 > 0 \). If \( p_2 \) never terminates in any extensions of \( \beta_0 \), we can find a set of executions with probability \( p_0 > 0 \) where \( p_2 \) never terminates, contradicting (*). Therefore, in some extension of \( \beta_0 \), \( p_2 \) terminates. Let \( \beta_1 \) be the finite execution that combines \( \beta_0 \) and this extension until \( p_0, p_1, p_2 \) terminate. We apply the same argument with execution \( \beta_1 \) and process \( p_3 \) to obtain a finite execution \( \beta_2 \) where \( p_0, \ldots, p_3 \) terminate. We continue this construction with all other processes, to obtain a finite execution \( \beta_{M-1} \) where all processes \( p_0, \ldots, p_M \) terminate.

Theorem 9 is generalized to the case of \( t = \lceil N/3 \rceil \) by the simulation of [21] as previously described for the deterministic algorithm.

8. CONCLUSIONS

Our results contribute to a better understanding of fault tolerance and efficiency of renaming in different settings. Namely, we showed that with the use of randomization renaming can be solved efficiently for any \( t < N \) under the non-rushing adversary. Our solution, presented in Alg. 1, terminates in \( O(\log N) \) rounds w.h.p., and works for any \( t < N \). We also showed that Alg. 1 works for \( t = 1 \) under the rushing adversary. For the general case of \( t > 1 \), we strengthened our algorithm with a cryptographic commitment scheme. An open problem is whether it is possible to avoid the use of cryptography when \( t > 1 \).

We then considered the order-preserving renaming and proved a \( t < N/3 \) bound for both deterministic and randomized protocols. This result reveals a separation between the resiliency of renaming and order-preserving renaming. It would be interesting to find minimum assumptions on the power of the adversary necessary to solve order-preserving renaming with \( t \geq N/3 \).

Acknowledgments

The authors are thankful to Henrique Moniz for his helpful comments on this work.

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/LA0021/2013, and via the project PEPITA (PTDC/EEI-SCR/2776/2012).

9. REFERENCES


