Distrib. Comput. (2016) 29:341–359
DOI 10.1007/s00446-016-0265-z

Randomized mutual exclusion on a multiple access channel

Marcin Bienkowski[1] · Marek Klonowski[2] · Miroslaw Korzeniowski[2] ·
Dariusz R. Kowalski[3]

Received: 14 September 2014 / Accepted: 19 January 2016 / Published online: 15 February 2016

Abstract In this paper we consider the mutual exclusion problem on a multiple access channel. Mutual exclusion is one of the fundamental problems in distributed computing. In the classic version of this problem, n processes execute a concurrent program that occasionally triggers some of them to use shared resources, such as memory, communication channel, device, etc. The goal is to design a distributed algorithm to control entries and exits to/from the shared resource (also called a critical section), in such a way that at any time, there is at most one process accessing it. In our considerations, the shared resource is the shared communication channel itself (multiple access channel), and the main challenge arises because the channel is also the only mean of communication between these processes. We consider both the classic and a slightly weaker version of mutual exclusion, called ε-mutual-exclusion, where for each period of a process staying in the critical section the probability that there is some other process in the critical section is at most ε. We show that there are channel settings, where the classic mutual exclusion is not feasible even for randomized algorithms, while the ε-mutual-exclusion is. In more relaxed channel settings, we prove an exponential gap between the makespan complexity of the classic mutual exclusion problem and its weaker ε-exclusion version. We also show how to guarantee fairness of mutual exclusion algorithms, i.e., that each process that wants to enter the critical section will eventually succeed.

Keywords Distributed algorithms · Multiple access channel · Mutual exclusion

Some preliminary results of this paper were published in the Proceedings of 27th International Symposium on Theoretical Aspects of Computer Science (STACS), 2010, pp. 83–94. Research Supported by Polish National Science Centre Grants DEC-2012/07/B/ST6/01534 and DEC-2013/09/B/ST6/01538, and by the Engineering and Physical Sciences Research Council [Grant Number EP/G023018/1].

✉ Marek Klonowski
Marek.Klonowski@pwr.wroc.pl;
Marek.Klonowski@pwr.edu.pl

[1] Institute of Computer Science, University of Wrocław, Wrocław, Poland

[2] Department of Computer Science, Faculty of Fundamental Problems of Technology, Wroclaw University of Technology, Wrocław, Poland

[3] Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK

1 Introduction

In this paper, we consider randomized algorithms for mutual exclusion: one of the fundamental problems in distributed computing. We assume that there are n different processes, each labeled by its unique identifier (ID) between 0 and $n-1$, communicating through a multiple access channel (MAC). The computation and communication proceed in synchronous slots, also called *rounds*. In the mutual exclusion problem, each process executes a concurrent program and occasionally requires exclusive access to shared resources. The part of the code corresponding to this exclusive access is called a *critical section*. The goal is to provide a mechanism that controls entering and exiting the critical section and guarantees exclusive access at any time. The main challenge is that the designed mechanism must be universal, in the sense that exclusive access must be guaranteed regardless of the times of access requests made by other processes.

Multiple Access Channel (MAC). We consider a multiple access channel as both communication medium and the shared-access device. As a communication medium, MAC allows each process either to transmit or listen to the channel at a round,[1] and moreover, if more than one process transmits, then a *collision* (signal interference) takes place.

Depending on the devices used in the system, there are several additional settings of MAC that need to be considered. One of them is the ability of a process to distinguish between background noise when no process transmits (also called *silence*) and collision. If such capability is present at each process, we call the model *with collision detection* (CD for short); if no process has such ability, then we call the setting *without collision detection* (no-CD). Another feature of the model is an access to the global clock (GC for short) by all processes or no such access by any of them (no-GC). In both GC and no-GC the system is synchronized and operates in rounds and the difference is that in GC processes know global numbering of rounds and in the no-GC they do not. That is, in the no-GC model processes set their local clocks to round 0 in the moment of the start of their own executions. The third parameter to be considered is the knowledge of the total number of available processes n (KN for short) or the lack of it (no-KN). In particular, in the KN model the algorithm executed by any process can explicitly use n. In the no-KN model the number of processes is always finite but can be arbitrary large.

Mutual Exclusion Problem. In this problem, each concurrent process executes a protocol partitioned into the following four sections.

Entry:	The part of the protocol executed in preparation for entering the critical section.
Critical:	The part of the protocol to be protected from concurrent execution.
Exit:	The part of the protocol executed on leaving the critical section.
Remainder:	The rest of the protocol.

These sections are executed cyclically in the order: remainder, entry, critical, and exit. Intuitively, the remainder section corresponds to local computations of a process, and the critical section corresponds to the access to the shared object (the channel in our case). Sections entry and exit are the parts that control switching between remainder and critical sections in a process, in order to ensure some desired properties of the whole system.

In the traditional mutual exclusion problem [1,2], in the context of shared-memory model, the adversary controls the sections remainder and critical. In particular, it controls their duration in each cycle, subject only to the obvious assumptions that this duration in each cycle is finite or the last performed section is the remainder one. The mutual exclusion algorithm, on the other hand, provides a protocol for the entry and exit sections of each process. Hence, the mutual exclusion problem can be seen as a game between the adversary controlling the lengths of remainder and critical sections of each process (each such section for each process may have different length) and the algorithm controlling entry and exit sections. The goal of the algorithm is to guarantee several useful properties of the execution (to be defined later), while the goal of the adversary is to prevent it. Note that the sections controlled by the adversary and those controlled by the algorithm are interleaved in the execution. It is typically assumed that any interference between sections controlled by the adversary and those of the algorithm is forbidden. In particular, no variable used by the algorithm, i.e., in the entry and exit sections, can be modified by the adversary in the critical and remainder sections, and vice versa.

In the model of communication over MAC, a process in the entry or the exit section can do the following in a single round:

– Perform some action on the channel (either transmit a message or listen).
– Do some local computation.
– Change its section either from entry to critical or from exit to remainder.

We assume that changing sections occurs momentarily between consecutive rounds, i.e., in each round a process is exactly in one section of the protocol.

Since a multiple-access channel is both the only communication medium and the exclusively shared object, additional constraints, different from the classic ones regarding, e.g., shared memory objects, must be imposed:

– No process in the remainder section is allowed to transmit on the channel.
– A process in the critical section has to transmit a message on the channel in each round until it moves to the exit section, and each such message must be labeled *critical*; we call such messages *critical messages*.

If any of these conditions was violated, the adversary would have an unlimited power of creating collisions on the channel. Let us recall that it is assumed that the adversary controls remainder and critical sections. Thus the adversary could impose arbitrary transmissions of processes staying in the remainder section and in effect prevent any communication. On the other hand, the process in critical section needs to inform other processes that the channel is busy.

A classic mutual exclusion algorithm should satisfy the following three properties for any round i of its execution.

[1] Most of the previous work on MAC, motivated by Ethernet applications, assumed that a process can transmit and listen simultaneously. Instead, our work follows the recent trends of wireless applications where such simultaneous activities are excluded due to physical constraints.

Exclusion: at most one process is in the *critical* section in round i.

Unobstructed exit: if a process p is in the exit section in round i, then process p will switch to the remainder section eventually after round i.

No deadlock: if there is a process in the entry section in round i, then *some* process will enter the critical section eventually after round i.

To strengthen the quality of service guaranteed by mutual exclusion algorithms, the following property—stronger than no-deadlock—has been considered:

No lockout: if a process p is in the entry section in round i, then process p itself will enter the critical section eventually after round i.

To some extent, this property ensures fairness: each process demanding access to the critical section will eventually get it.

As we show, in some model settings the exclusion condition is impossible or very costly to achieve. Therefore, in this paper we introduce and investigate a slightly weaker condition:

ε-exclusion: for every process p and for every time interval in which p is continuously in the critical section, the probability that in any round of this time interval there is another process being in the critical section is at most ε.

Intuitively, ε-exclusion guarantees mutual exclusion "locally", i.e., for every single execution of the critical section by a process, with probability at least $1 - \varepsilon$. The version of the problem satisfying ε-exclusion condition is called *ε-mutual-exclusion*.

Complexity Measure. We use the *makespan* measure, as defined by Czyzowicz et al. [3], in the context of deterministic algorithms. The makespan of an execution of a given deterministic mutual exclusion algorithm is defined as the maximum length of a time interval in which there is some process in the entry section and there is no process in the critical section. Taking maximum of such values over all possible executions defines the makespan of the algorithm. In order to define expected makespan, suitable for randomized algorithms considered in this work, we need more formal definitions of an adversarial strategy. Let $\mathscr{P}$ be a strategy of the adversary, defined as a set of n sequences, where each sequence corresponds to a different process and contains, subsequently interleaved, lengths of remainder and critical sections of the corresponding process. We assume that each sequence is either infinite or of even length; the latter condition means that after the last critical section the corresponding process runs the remainder section forever. For a given mutual exclusion algorithm $\textsc{Alg}$ and adversarial strategy $\mathscr{P}$, we define $L(\textsc{Alg}, \mathscr{P})$ as a random variable equal to the maximum length of a contiguous time interval in which there is some process in the entry section and there is no process in the critical section in an execution of $\textsc{Alg}$ run against fixed strategy $\mathscr{P}$. The expected makespan of algorithm $\textsc{Alg}$ is defined as the maximum of expected values of $L(\textsc{Alg}, \mathscr{P})$, taken over all adversarial strategies $\mathscr{P}$. Note that every algorithm with expected makespan bounded for all executions satisfies the no-deadlock property with probability 1, but not necessarily no-lockout.

For the ε-mutual-exclusion problem, defining makespan is a bit more subtle. We call an execution *admissible* if the mutual exclusion property is always fulfilled, i.e., no two processes are in the critical section in the same round. Then, in the computation of the (expected) makespan, we neglect non-admissible executions.

1.1 Previous and related work

The multiple access channel is a well-studied model of communication. In many problems considered in this setting, one of the most important issues is to ensure that successful transmissions occur in the computation. These problems are often called *selection problems*. They differ from the mutual exclusion problem by the fact that they focus on successful transmissions within a bounded length period, while mutual exclusion provides control mechanism for dynamic and possibly unbounded executions. In particular, it includes recovering from long periods of cumulative requests for the critical section as well as from long periods containing no request. Additionally, selection problems were considered typically in the context of the Ethernet or combinatorial group testing, and as such they allowed a process to transmit and to listen simultaneously, which is not the case in our model motivated by wireless applications. Selection problems can be further split into two categories. In the static selection problems, it is assumed that a subset of processes become active at the same time and a subset of them must eventually transmit successfully. Several scenarios and model settings, including parameters considered in this work, such as CD/no-CD, GC/no-GC, KN/no-KN, randomization/determinism, were considered in this context [4–13]. In the wake-up problem, processes are awaken in (possibly) different rounds and the goal is to ensure that there will be a round with successful transmission ("awakening" the whole channel) shortly after the first process is awaken [14–17].

More dynamic kinds of problems, such as transmission of dynamically arriving packets, were also considered in the context of MAC. In the (dynamic) packet transmission problem, the aim is to obtain bounded throughput and bounded latency. Two models of packet arrival were considered: stochastic [18] and adversarial queuing [19,20]. There are two substantial differences between these settings and our work. First, the adversaries imposing dynamic packet arrival are different from the adversary simulating execution of concurrent protocol. Second, as already mentioned in the context of selection problems, these papers were inspired by the Ethernet applications where it is typically allowed to transmit and listen simultaneously.

In the context of deterministic algorithms for MAC under the CD/GC/KN setting, that when none of these three characteristics is available, mutual exclusion is infeasible [3]. Moreover, the authors of [3] presented an optimal—in terms of the makespan measure—$O(\log n)$ rounds algorithm for the model with CD. They also developed algorithms achieving makespan $O(n \log^2 n)$ in the models with GC or KN only, which, in view of the lower bound $\Omega(n)$ on deterministic solutions proved for any model with no-CD, is close to optimal.

1.2 Our results

We consider the mutual exclusion problem in the multiple access channel, and—for the sake of efficiency—we introduce its weaker version: the ε-mutual-exclusion problem.

We extend the results of [3] (which guaranteed only the no-deadlock property) to focus on fairness, ensuring the no-lockout property. Additionally, in contrast to the previous work on mutual exclusion on MAC, we also study randomized solutions. In the case of the mutual exclusion problem, we allow randomized algorithms to have variable execution time but they have to be always correct. On the other hand, a randomized solution for the ε-mutual-exclusion problem is allowed to err with some small probability ε. Thus, for the former problem, we require a Las Vegas type of solution, whereas for the latter we admit Monte Carlo algorithms. Note that a very small (e.g., comparable with probability of hardware failure) risk of failure (i.e., situation wherein two or more processes are in the critical section at the same round) is negligible from a practioner's point of view. Below, we describe our results when only the no-deadlock property is required (they are summarized in Table 1); later we show how to extend them to achieve the no-lockout property.

We show that for the most severe channel setting, i.e., no-CD, no-GC and no-KN, mutual exclusion is not feasible even for randomized algorithms (cf. Sect. 2), thus extending the analogous result of [3], holding for deterministic algorithms. On the other hand, we show that the ε-mutual-exclusion problem in this setting can be solved quite efficiently, by constructing an IFS+ algorithm with expected makespan $O(\log n \cdot \log(1/\varepsilon))$ (cf. Sect. 3.1). Our algorithm derives its name and some core ideas from the *Increase From Square* (IFS) algorithm [16] for the wake-up problem.

In a more relaxed setting, we prove an exponential gap between the complexity of the mutual exclusion problem and the ε-mutual-exclusion problem. Specifically, we show that the expected makespan of (randomized) solutions for the mutual exclusion problem in the no-CD setting is $\Omega(n)$, even if the algorithm knows n and has access to the global clock (cf. Sect. 2). On the other hand, algorithm IFS+ clearly also works in this setting

Another exponential gap occurs when collision detection is available. Although we show that the makespan of any randomized mutual exclusion algorithm is at least $\Omega(\log n)$ (cf. Sect. 2), we construct an algorithm for the ε-mutual-exclusion problem with expected makespan $O(\log \log n + \log(1/\varepsilon))$ (cf. Sect. 3.2). Note that the algorithm neither uses the knowledge of n nor the global clock, whereas the lower bound holds even in presence of these channel capabilities.

Finally, we present a generic deterministic method that takes a mutual exclusion algorithm with the no-deadlock property and turns it into one satisfying the stronger, no-lockout condition (cf. Sect. 4). Our scheme requires either collision detection capability or the knowledge of n, and increases the makespan of the original algorithm by an additive term of $O(\log n)$. This method applied to the deterministic algorithms from [3] produces efficient deterministic solutions satisfying the no-lockout property and applied to our randomized algorithms allows us to guarantee the no-lockout property for all our algorithms, in most cases without increasing their asymptotic complexity.

Note that consecutive entry and exit sections of a process can be potentially related. In particular, a process can execute a different code each time it enters its entry (or exit) section. While we allow such behaviour in general (for instance our

Table 1 Summary of our and previous results for the no-deadlock variant of the problem

	Mutual exclusion (deterministic)	Mutual exclusion (randomized)	ε-mutual-exclusion (randomized)
No-CD, no-GC, no-KN	Infeasible [3]	Infeasible (Theorem 1)	$O(\log n \cdot \log(1/\varepsilon))$ (Theorem 4)
No-CD, only GC or KN	$O(n \log^2 n)$, $\Omega(n)$ [3]	$\Omega(n)$ (Theorem 3)	
CD	$\Theta(\log n)$ [3]	$\Omega(\log n)$ (Theorem 2)	$O(\log \log n + \log(1/\varepsilon))$ (Theorem 5)

The complexity measure is the (expected) makespan lower bounds hold for such general case), we use it in a very limited way in our algorithms. In particular, our algorithms that guarantee no deadlock (cf. Sect. 3) use the same code in every execution of entry section, while our algorithms that additionally guarantee the no-lockout property (cf. Sect. 4) use only counters that are preserved between consecutive exit sections.

Relating our setting and results to the previously considered ones. There are many similar problems in distributed computing that employ choosing exactly one entity (out of many) while the communication is strictly restricted. The closest examples are the problems of choosing a leader, clock synchronization and wake-up on a single hop radio network. Although in construction of our algorithms we use some techniques developed in the context of these problems, they needed to be significantly modified. For example, in Sect. 3.2, we simulate the Willard's algorithm [13], but we need to introduce modifications to overcome problems caused by the more demanding nature of our setting. In particular, in our scenario, we have to cope with the dynamic nature of the mutual exclusion problem, where new processes can be activated on the fly and the lack of mechanism acknowledging the right to enter the critical section.

Since our model is more constrained and complex than the previously considered models of the collision channel, one may be tempted to use lower bounds proved for related settings in order to obtain interesting limitations in our settings. This approach could be however insufficient, due to a more demanding nature of the mutual exclusion problem. For example, for the mutual exclusion problem with n processes and CD we need $\Omega(\log n)$ rounds as proved in Theorem 2. On the other hand, the leader election problem in single hop radio networks with CD has a lower bound of only $\Omega(\log\log n)$ (that can be matched by an algorithm). That is, solving the mutual exclusion problem in our setting is substantially more difficult then choosing a leader in a single hop radio network. Another example is an impossibility result for the weakest model of the channel presented in this work—to the best of our knowledge, there are no prior results of this type so far.

A more relaxed version of the mutual exclusion problem—the ε-mutual-exclusion problem, introduced in this paper—is closer to the previously considered problems on the collision channel. One may suspect that some existing lower bounds for the leader election or wake-up problems can be somehow transferred to the new model. In this work, we do not discuss the lower bounds for ε-mutual-exclusion problem, focusing rather on the issue how this relaxation of the mutual exclusion problem improves the complexity. We leave an investigation of potential relations between this and other similar problems as a future work.

2 Lower bounds for the mutual exclusion problem

In our lower bounds, we use the concept of transmission schedules to capture transmission/listening activity of processes in the entry or the exit section. Transmission schedule of a process p can be regarded as a binary sequence π_p describing the subsequent communication actions of the process, up to the point when it enters the critical section. The sequence can be finite or infinite. For non-negative integer i, $\pi_p(i) = 1$ means that process p transmits in round i after starting its current section, while $\pi_p(i) = 0$ means that the process listens in round i. We assume that round 0 is the round in which the process starts its current run of the entry or the exit section.

The following results extend the lower bounds and impossibility results for deterministic mutual exclusion proved in [3] to randomized solutions. All the presented lower bounds apply even if we do not require no-lockout, but the weaker no-deadlock property.

Theorem 1 *There is no randomized mutual exclusion algorithm with no-deadlock property holding with a positive probability in the setting without collision detection, without global clock and without knowledge of the number n of processes.*

Proof Suppose, for contradiction, that there exists a mutual exclusion algorithm $\mathscr{R}$ in the considered setting that accomplishes no-deadlock with a positive probability for each adversarial schedule. Our goal is to show that, under this assumption, there exists an execution violating the mutual exclusion.

Since algorithm $\mathscr{R}$ does not have the parameter n in its input, the adversary may consider any finite subset of processes with IDs being non-negative integers, and decide on n being the maximum of used IDs. For a process p, let $\mathbb{F}_p$ be the set of all executions that occur with positive probability of the first entry section of algorithm $\mathscr{R}$ by a process p, under the scenario where only process p is in this section and all other processes are in the remainder section (i.e., this is the first entry section in the global execution and no other process tries to enter). Note that during each execution in $\mathbb{F}_p$, process p hears only background noise from the channel whenever it listens and, by non-deadlock property, it eventually enters the critical section (in a finite time). Let $\mathscr{E}_p \in \mathbb{F}_p$ be the execution where p enters the critical section, and let π_p be the transmission schedule of process p during $\mathscr{E}_p$. Let $|\pi_p|$ be the length of execution $\mathscr{E}_p$.

Consider a family of transmission sequences π_p over all processes p. (Recall that we consider all possible non-negative integer ids of processors at this point of the proof.) Note that transmission sequences may have different lengths, but all of them are bounded. We call a transmission sequence *proper* if there is at least one occurrence of 1 in it. It is easy to see that all sequences π_p, except at most one, are proper. Otherwise, the adversary could choose two non-proper sequences π_p, π_q and build the following execution contradicting the mutual exclusion property. It starts the first entry sections of p and q in rounds $|\pi_q| + 1$ and $|\pi_p| + 1$, respectively. By inductive argument on the number of rounds, we can extend transmission sequences of p, q in such a way that there is no transmission. This is because the previously built prefixes of these sequences and the feedback from the channel make the execution undistinguishable from the previously defined executions $\mathscr{E}_p, \mathscr{E}_q$ at process p, q, respectively; therefore, as in $\mathscr{E}_p, \mathscr{E}_q$, there is a way that the algorithm chooses not to transmit in both p and q. This may be continued inductively up to round $|\pi_p| + |\pi_q|$, where, again by the fact that the built execution is the same in p, q as in executions $\mathscr{E}_p, \mathscr{E}_q$, respectively, and because the built transmission sequences are exactly π_p, π_q. Hence, at most one sequence is not proper.

Consider the following adversarial strategy. Let q denote the ID of a non-proper transmission sequence, if it exists, or 0 otherwise. For every process p, where $p > q + 1$, we define ℓ_p to be the position of the first 1 in sequence π_p. Let $a = \max\{|\pi_q|, |\pi_{q+1}|\}$ and let $b = \max\{\ell_p : q + 1 < p \leq q + 2a + 1\}$. For a process p, where $q + 1 < p \leq q + 2a + 1$, the adversary sets the length of the first remainder section to $b - \ell_p + \lceil (p - q - 1)/2 \rceil$, and processes $q, q + 1$ are set to start their first remainder sections in rounds $b + a - |\pi_q|$ and $b + a - |\pi_{q+1}|$, respectively. Note that all these lengths are non-negative integers, and the starting points of processes $q, q + 1$ are after round b.

We show by induction on round $0 \leq i \leq b + a$ that there is an execution where no message is heard in all rounds from 1 to i. More precisely, we will be showing that there is an execution $\mathscr{E}$ where, under the defined adversarial schedule, all considered processes p follow their transmission schedules π_p up to round i of the execution and in all these rounds no message is heard on the channel. (This global round i may correspond to different positions in the transmission schedules.) Note that we assumed no collision detection, therefore the invariant implies that the same noise is heard during all these rounds. The invariant for $i = 0$ is true since no process has started its entry section yet, and no one has transmitted yet. Suppose that the invariant holds for some $0 \leq i < b + a$, we show it for round $i + 1$.

First, consider the case $i + 1 \leq b$. By the invariant for i, we get that from the point of view of a single process p, the execution $\mathscr{E}$ by the end of round i is the same as the corresponding prefix of execution $\mathscr{E}_p$ (in which only process p is in the entry section), and in both executions the transmission schedule of p is the same prefix of π_p. Note that this prefix can be extended by one more position according to the transmission schedule π_p when considered in execution $\mathscr{E}_p$, since the length of π_p is at least ℓ_p while the currently built prefix of π_p has length

$$i - \left(b - \ell_p + \left\lceil \frac{p - q - 1}{2} \right\rceil\right) < b - \left(b - \ell_p + \left\lceil \frac{p - q - 1}{2} \right\rceil\right) \leq \ell_p \; . \tag{1}$$

It follows that in both executions $\mathscr{E}, \mathscr{E}_p$ the transmission sequences of process p up to round $i + 1$ can be made the same, and they are a prefix of schedule π_p. It remains to show that in round $i + 1$ the noise is heard on the channel. This follows again from relation (1), which guarantees that the first transmission of any process $p > q + 1$, following its transmission sequence π_p, occurs after round b of the execution, and processes $q, q + 1$ start their entry sections after round b.

Next, consider the case $b < i + 1 \leq b + a$. The same argument as in the previous case justifies that, because of the noise on the channel heard in all previous rounds and the similarity to the executions $\mathscr{E}_p$, all prefixes of the transmission sequences π_p built by round i in execution $\mathscr{E}$ can be extended by one more position according to π_p. To ensure that the noise is heard also in round $i + 1$, consider processes with IDs $p = q + 2 \cdot (i + 1 - b)$ and $r = q + 2 \cdot (i + 1 - b) + 1$. Note that both IDs are bigger than $q + 1$ and smaller or equal to $q + 2|\pi_q| + 1$. Moreover, since they start their entry sections just after rounds $b - \ell_p + \lceil (p - q - 1)/2 \rceil = i + 1 - \ell_p$ and $b - \ell_r + \lceil (r - q - 1)/2 \rceil = i + 1 - \ell_r$, respectively, they both have the first value 1 in their schedules while being in round $i + 1$ (being prefixes of π_p, π_r, resp., of length ℓ_p, ℓ_r, resp.), and thus both transmit in this round. Hence the noise (collision) is heard. Recall that this noise is not distinguishable form the one caused by silence (no transmission), as there is no collision detection capability.

By the invariant for round $b + a$, the constructed execution $\mathscr{E}$ is not distinguishable from execution $\mathscr{E}_q$ from the point of view of process q, and similarly $\mathscr{E}$ is not distinguishable from $\mathscr{E}_{q+1}$ at process $q + 1$. Moreover, the first of them has length $(b + a) - (b + a - |\pi_q|) = |\pi_q|$ and the second one has length $(b + a) - (b + a - |\pi_{q+1}|) = |\pi_{q+1}|$. Therefore, each of the processes $q, q + 1$ chooses to enter the critical section at this point, as it could do in $\mathscr{E}_q, \mathscr{E}_{q+1}$, respectively. This violates mutual exclusion in the execution $\mathscr{E}$. □

Theorem 2 *The expected makespan of any randomized mutual exclusion algorithm is at least* $\log n$, *even in the setting with collision detection, with global clock and with knowledge of the number n of processes.*

Proof Suppose, for the sake of contradiction, that there is a mutual exclusion algorithm $\mathscr{R}$ whose expected makespan is smaller than $\log n$. Our goal is to show that, under this assumption, there exists an execution violating mutual exclusion.

For a process p, let $\mathbb{F}_p$ be the set of all possible executions of the first entry section of algorithm $\mathscr{R}$ by process p under the assumption that it starts its first entry section in global round 1 and there is no other process starting within the first $\log n$ rounds. Note that during each execution in $\mathbb{F}_p$ process p hears only background noise (i.e., silence) from the channel when listening. Therefore, by the probabilistic method, there is an execution $\mathscr{E}_p$ in the set $\mathbb{F}_p$ where process p enters the critical section within the first $\log n - 1$ rounds. Let π_p be the transmission schedule of process p during $\mathscr{E}_p$.

Consider sequences π_p of all processes $0 \le p < n$. There are less than n different 0-1 sequences of length at most $\log n - 1$, hence some two processes p, q have the same sequences $\pi_p = \pi_q$. We construct an execution $\mathscr{E}$ contradicting the mutual exclusion property as follows. The adversary starts the first entry sections for these two processes p, q in the very first round, while delaying other (they remain in the remainder section) till round $\log n$. Before round 1 of the execution, process p cannot distinguish it from $\mathscr{E}_p$, therefore it may decide to do the same as in $\mathscr{E}_p$, i.e., to set its first position of transmission schedule to $\pi_p(1)$. The analogous argument holds for process q, with respect to executions $\mathscr{E}$, $\mathscr{E}_q$ and the transmission schedule π_q. If this happens, either both processes transmit or both listen, which results in either no feedback (nobody listens) or silence heard (nobody transmits). This is caused by $\pi_p = \pi_q$ and the fact that a process cannot transmit and listen simultaneously in a round. This construction and output of the first round can be inductively extended up to round $|\pi_p| = |\pi_q|$, since from the point of view of process p (or q) the previously constructed prefix of $\mathscr{E}$ is not distinguishable from the corresponding prefix of execution $\mathscr{E}_p$ (or $\mathscr{E}_q$, respectively); indeed, the transmission schedules are the same and the feedback from the channel is silence whenever the process listens. Finally, by the very same reason, at the end of round $|\pi_p|$ both p and q are allowed to do the same as in $\mathscr{E}_p$ and $\mathscr{E}_q$, respectively, that is, to enter the critical section. This violates the mutual exclusion property that should hold for $\mathscr{E}$. □

Theorem 3 *The expected makespan of any randomized mutual exclusion algorithm is at least $n/2$ in the absence of collision detection capability, even in the setting with global clock and with knowledge of the number n of processes.*

Proof To arrive at a contradiction, let $\mathscr{R}$ be a randomized mutual exclusion algorithm, whose expected makespan is c, where $c < n/2$. We show that there exists an execution violating the mutual exclusion property.

For a process p, let $\mathbb{F}_p$ be the set of all possible executions of the first entry section of algorithm $\mathscr{R}$ by process p under the assumption that it starts its first entry section in the global round 1 and there is no other process starting within the first $n/2$ rounds. Note that during each execution in $\mathbb{F}_p$ process p hears only noise (i.e., silence or collision, which are indistinguishable due to the lack of collision detection) from the channel when listening. Therefore, by the probabilistic method, there is an execution $\mathscr{E}_p \in \mathbb{F}_p$, where process p enters the critical section within the first $n/2 - 1$ rounds. Let π_p be the transmission schedule of process p during $\mathscr{E}_p$.

Consider sequences π_p of all processes $0 \le p < n$. We construct an execution $\mathscr{E}$ contradicting the mutual exclusion property as follows. First, we select a set P^* of processes that start their first entry sections in round 1, while the others stay in the remainder section till at least round $n/2$. To this end, we create a sequence $\{P_j\}_j$ starting from a set $P_0 = \{0, \dots, n-1\}$, and we define two operations on set P_j of processors.

- *Remove shortest*: from set P_j remove the (unique) processor p with the shortest transmission schedule, i.e., such that $\exists_{i \in [1, n/2-1]} \left(|\pi_p| = i \ \& \ \forall_{q \in P_j, q \ne p} |\pi_q| > i \right)$.
- *Remove transmitting*: from set P_j remove a processor p, for which there exists a round, where only p transmits, i.e., such that $\exists_{i \in [1, n/2-1]} \left(\pi_p(i) = 1 \ \& \ \forall_{q \in P_j, q \ne p} \pi_q(i) = 0 \right)$.

Now, we proceed inductively: P_{j+1} is created from P_j by applying any of these operations. If at some round neither of these operations can be applied, then we call the resulting set P^*. Observe that each of the operations above can be applied at most $n/2 - 1$ times (i.e., at most once per each $i \in [1, n/2 - 1]$), which results in removing of at most $2 \cdot (n/2 - 1) = n - 2$ processes from set P; hence $|P^*| \ge 2$. Having subset P^* of processes, the adversary starts first entry sections for all processes in P^* in the very first round, while it delays others (they remain in the remainder section) until round $n/2$. Note that before round 1 of the constructed execution $\mathscr{E}$, a process $p \in P^*$ cannot distinguish $\mathscr{E}$ from $\mathscr{E}_p$, therefore it may decide to do the same as in $\mathscr{E}_p$, i.e., to set its first position of transmission schedule to $\pi_p(1)$. This happens for all processes in P^*. Since there is no single transmitter in round 1 (otherwise the operation *remove transmitting* could have been applied), all listening processes hear the noise (recall that silence is not distinguishable from collision in the considered setting).

This construction and the output of the first round can be inductively extended up to round $|\pi_p|$, where $p \in P^*$ is a process with the shortest schedule π_p among processes in P^*. This is because from the point of view of a process $q \in P^*$ the previously constructed prefix of $\mathscr{E}$ is not distinguishable from the corresponding prefix of execution $\mathscr{E}_q$; indeed, the transmission schedules are the same and the feedback from the channel is silence whenever the process listens. Finally, observe that there exists (at least one) processor q, such that $|\pi_q| = |\pi_p|$ (otherwise the operation *remove shortest* could have been applied). At the end of round $|\pi_p|$, both p and q are allowed to do in $\mathscr{E}$ the same action as in $\mathscr{E}_q$, that is, to enter the critical section. This violates the exclusion property that should hold for the constructed execution $\mathscr{E}$. □

3 Algorithms for the ε-mutual-exclusion problem

In this section, we present two randomized algorithms solving the ε-mutual-exclusion problem for various channel capabilities. Namely, we present an algorithm for the no-CD variant and an algorithm for the CD variant of the problem. No algorithm requires other channel capabilities, such as the knowledge of n or the global clock. These algorithms work solely in entry sections, i.e., their exit sections are empty, and they guarantee only the no-deadlock property. Later, in Sect. 4, we show how to add exit section subroutines to all our algorithms in order to guarantee the no-lockout property while keeping the makespan bounded. In our algorithms, we extend some techniques developed in the context of other related problems, such as the wake-up problem [16] and the leader election problem [13].

Throughout this section, we use the following notation and assumptions. We assume that processes listen in these rounds in which they do not transmit. We say that there is a *successful transmission* in a given round if in this round one process transmits and others listen. By saying that a process *resigns* (or alternatively *changes state to resigned*), we mean that it will not try to enter the critical section and will not attempt to transmit anything until another process starts the exit section. (Recall that a process in a critical section transmits a critical message in each round.) Our algorithms are memoryless in the following sense: a resigned process will restart its protocol from scratch after the first round when it does not hear a critical message.

3.1 Collision detection not available

In the first variant, we do not assume any channel capabilities, i.e., we consider no-CD, no-KN, and no-GC version. In our construction, we build on the ideas from algorithm *Increase From Square* (IFS) [16], which efficiently solves the wake-up problem for the considered variant of channel capabilities. In the wake-up problem, each process is activated in an arbitrary round by the adversary and the goal is to have a round with a successful transmission as quickly as possible.

A rough idea behind IFS [16] is that process i starts with probability $q_i = \Theta(1/i^2)$ and operates in phases of length $k = \Theta(\log(1/\varepsilon))$. In any round of each phase, process i transmits a message with probability q_i and at the end of the phase, q_i is doubled until it exceeds $1/2$. In the original proof, it is argued that for any pattern of adversarial activations, after $O(\log n)$ phases, there will be k consecutive rounds where the sum of probabilities of all active processes is between $1/2$ and $3/2$. This ensures a successful transmission occurring with probability $1-\varepsilon$ in one of these rounds.

3.1.1 *Problems with transforming wake-up into mutual exclusion*

The solution for the wake-up problem might seem to be a perfect candidate for an entry section algorithm: whenever a process enters its entry section, it starts to execute algorithm IFS and the single process that transmits successfully enters the critical section. Unfortunately, such an approach is far from being correct. One of the main problems is that the process that succeeded may not be aware of it being successful. (Recall that we assume that the process cannot transmit and listen simultaneously.) We could alleviate this problem by employing the following "killing rule": any process that hears a successful transmission from process j resigns and process j enters the critical section upon ending its IFS routine. In this informal description, we call such j a temporary leader. Note, however, that the period between a successful transmission of process j and the end of its IFS execution can be quite long. (This happens for example when there are many processes with small transmission probabilities: one of them may transmit successfully although its transmission probability is very low.) In such period, other processes with smaller IDs (and hence larger transmission probabilities) may have a better chance to start their entry sections. It is therefore quite likely that they could force j to resign and one of them would become a temporary leader. As changing the temporary leader this way can occur many times, it is not clear how to bound the makespan of such algorithm.

In order to be able to benefit from using such a "killing" process, we have to introduce an additional acknowledgement scheme. Namely, all processes try to send an announcing message first. We may expect the first successful transmission (say, by process j) when the sum of transmission probabilities is constant. Afterwards, all other processes that heard such announcing message try to transmit acknowledgement for j.

As soon as j hears such an acknowledgement, it enters the critical section. Such acknowledging scheme works well provided the sum of transmission probabilities of acknowledging processes is also appropriately large. In the problematic case, the sum of transmission probabilities of the acknowledging processes is small and they might not succeed in sending an acknowledgement. However, this implies that the transmission probability of process j is quite large. In such case we apply another "killing rule": if the transmission probability of a process is quite high, instead of sending an announcing message, it sends a *killing* message, which forces all remaining processes to resign. Note that this time process j does not need an acknowledgement, as the end of its IFS routine will occur soon.

To sum up, in our entry sections, we use a modified version of the IFS routine, combining the aforementioned acknowledging scheme and a "killing rule"; its precise description is given in the next section. While keeping the general framework of phases and the paradigm of increasing transmission probabilities, we introduce several substantial changes, such as a silence period at the beginning, three types of non-critical messages: *announcing*, *killing*, and *acknowledging*, and some minor adjustments of constants.

3.1.2 The IFS+ routine

The IFS+ routine executed by each process in its entry section is defined as follows. Let $k = 4 \cdot \lceil \log(8/\varepsilon)/\log(56/55) \rceil$, $y = \pi^2/6$, and $f_i = \lceil \log(y \cdot (i+1)^2) \rceil + 3$. Note that acting of each process depends on its ID, however we do not use the knowledge of the upper bound on the number of IDs (parameter n). Routine IFS+ for process i consists of $f_i + 3$ phases numbered from 0, each consisting of k rounds. In the first three phases (called *silent*), process i sets its *transmission probability* q_i to 0. In the h-th phase (for $h \geq 3$), process i sets *transmission probability* to $q_i = 2^{h-3-f_i}$. In other words, neglecting the three silent phases, the transmission probability is $1/2^{f_i}$ in the first non-silent phase, doubles between two consecutive phases, and is $1/2$ in the last phase. We write $q_i(t)$ if we want to emphasize that we use the value of q_i from a particular round t. Note that the runtime of the uninterrupted IFS+ routine of process i is fixed and equals to $k \cdot (f_i + 3) = O(\log n \cdot \log(1/\varepsilon))$. In our analysis, we do not aim at minimizing the constants, but rather at the proof simplicity.

The processes behave in the following way. A process in the entry section can be in any of the following four states: *announcing*, *acknowledging(j)*, *killing*, or *resigned*, where the second one is parameterized with a number j being an ID of another process. We use the phrase *resigns* as a synonym for *changes state to resigned*. All state transitions occur instantly between rounds, i.e., in any round a process is exactly in one state. That said, when we write that a process resigns in round t, we mean that it changes its state to resigned between round t and $t+1$. In our pseudocode, we additionally use the *critical-ready* state: a process that is assigned this state, enters the critical section in the next round.

Each process starts its entry section in the announcing state and starts its IFS+ routine. Recall that each phase of the IFS+ routine consists of k rounds. These rounds are partitioned into, interleaved, $k/2$ odd-numbered and $k/2$ even-numbered ones. A process that is in its entry section and has not resigned yet is called *active*; sometimes we use the term *becomes active* in place of *starts its entry section*. An active process j transmits its current state in each odd round with transmission probability q_j. (The transmission events for particular processes are independent in each odd round.) In an even round, process j does not transmit anything independently of q_i. The necessity of these silent even-numbered rounds will become clear later. Note that this odd-even distinction applies only to entry sections: in the critical section, a process consistently transmits a critical message in each round.

An active process changes its state between rounds t and $t+1$ based on received message (if any) and local time. First, it can change its state because there was a successful transmission in round t, namely:

(R1) A process hearing a critical or killing message resigns;
(R2) A killing process hearing an announcing message does nothing;
(R3) A non-killing process hearing an announcing message from j changes state to acknowledging(j);
(R4) A process j hearing an acknowledging(j) message enters the critical section (i.e., its critical section starts in round $t+1$);
(R5) A process j hearing an acknowledging(ℓ) message, where $\ell \neq j$, resigns.

Between rounds t and $t+1$, after rules (R1)–(R5) are applied, a process which is still active and not going to enter the critical section in the next round may still change its state if any of the following two time-related rules is triggered:

(T1) An announcing process which—in round $t+1$—starts its last phase of the IFS+ routine changes its state to killing;
(T2) An active process that finished its IFS+ routine enters the critical section.

Note that if a process ends its IFS+ routine (i.e., executes rule (T2)), then it has to be either in killing or in acknowledging state. In particular, an announcing process cannot finish its IFS+ routine, as this would mean that it was announcing at the end of the second-to-last phase and thus it would have changed its state to the killing state in the beginning of the last phase. (Here we implicitly used the fact that, based on the transition rules (R1)–(R5) and (T1)–(T2), there is no return to the announcing state once changing it to any other state.) The complete pseudocode for entry section is given in Algorithm 1.

As mentioned already, a resigned process just waits for a critical section to occur, and then it changes its state to announcing and restarts IFS+ routine as soon as it learns that the critical section has finished.

Algorithm 1 Entry section for process i

```
Initialization:
  $k \leftarrow 4 \cdot \lceil \log(8/\varepsilon)/\log(56/55) \rceil$
  $y \leftarrow \pi^2/6$
  $f_i \leftarrow \lceil \log(y \cdot (i+1)^2) \rceil + 3$
  STATE ← announcing

Actions:    for round $\tau \in \{1, \ldots, k\}$ of phase $h \in \{0, \ldots, f_i + 2\}$
  if $h < 3$ then $q_i \leftarrow 0$ else $q_i \leftarrow 2^{h-3-f_i}$
  if $k$ is odd and STATE ≠ resigned then
      transmit STATE with probability $q_i$
  if STATE ≠ resigned and process received a message then
      $j \leftarrow$ ID of transmitting process
      RECV ← received message (state of $j$)
      if RECV = critical or RECV = killing then                ▷ (R1)
          STATE ← resigned
      if RECV = announcing then                                ▷ (R2),(R3)
          if STATE ≠ killing then
              STATE ← acknowledging($j$)
      if RECV = acknowledging($\ell$) then                     ▷ (R4),(R5)
          if $\ell = i$ then
              STATE ← critical-ready
          else
              STATE ← resigned
  if STATE = announcing and $\tau = k$ and $h = f_i + 1$ then   ▷ (T1)
      STATE ← killing
  if STATE ≠ resigned and $\tau = k$ and $h = f_i + 2$ then     ▷ (T2)
      STATE ← critical-ready
  if STATE = critical-ready then
      enter critical section in the next round
```

3.1.3 Structural properties of IFS+

We analyze our election mechanism, i.e., algorithm IFS+, for a single critical section. In particular, in the analysis, we consider only rounds after the previous critical section (if any), as all IFS+ routines (if any) are restarted right after the critical section. By τ_{start} we denote the first round after the previous critical section, in which there is a process starting its IFS+ routine. Hence, without loss of generality, we assume that processes start their IFS+ routines in round τ_{start} or later and the starting points are chosen by the adversary in an arbitrary manner. For any process i, we denote its starting round by t_i (t_i can be also infinite if the process does not become active after round τ_{start}). We call all rounds starting from τ_{start} and ending at the last round before some process enters the critical section *election rounds*.

We observe that any process j that enters the critical section transmits a critical message, and all processes listening in this round resign by rule (R1). Thus, for our algorithm, the necessary and sufficient condition for the mutual exclusion is that at a round when j enters the critical section, no other process enters the critical section and no other process transmits anything. For short, in such case, we say that j starts the critical section *with mutex*.

The first lemma states that processes in acknowledging states always certify the existence of some single alive process.

Lemma 1 *At any election round, the following two properties hold.*

1. *There is at least one process in the announcing or killing state.*
2. *If there are some acknowledging processes, then they are all in the same state acknowledging(j), where process j is announcing or killing.*

Proof We show the lemma by a simple induction. The statement holds at τ_{start} as there is at least one announcing process in that round (the process that starts its IFS+ routine) and there are no processes in acknowledging states.

Assume now that both properties hold in round t and we show them for round $t+1$. We may neglect the processes that start their entry sections (and IFS+ routines) in round $t+1$, as they start in announcing states and this may only help in preserving the properties. We divide the change of processes' states into two parts: in the first one we check the effect of applying rules (R1)–(R5) and in the second one—the rules (T1)–(T2).

For the rules (R1)–(R5) to apply, there has to be a successful transmission in round t. Let i be the transmitting process. We consider several cases.

- Process i was announcing. By rule (R3), all other processes in announcing or acknowledging states change their state to acknowledging(i). By rule (R2), all killing processes remain intact.
- Process i was killing. By rule (R1), all other processes resign.
- Process i was in state acknowledging(j). Then by the inductive assumption, j was announcing or killing in round t, and therefore, by rule (R4), j would enter the critical section at time $t+1$. In this case lemma holds emptily as $t+1$ is not an election round.

We showed that the first two cases preserve the properties of the lemma and the third one cannot occur. Independently of the successful transmission in round t, rules (T1)–(T2) might be triggered for some processes. However, rule (T1) does not change the validity of the conditions of the lemma and rule (T2) would make round $t+1$ a non-election round. This concludes the inductive proof of the lemma. □

By the rules of the algorithm IFS+, there are two possibilities that a process enters the critical section: either this is triggered by rule (R4) (a successful transmission of an acknowledging message) or the process ends its IFS+ routine and enters the critical section by rule (T2). In the lemma below, we focus on the former case.

Lemma 2 *If a process enters the critical section before the end of its IFS+ routine, then the mutual exclusion property is satisfied.*

Proof The second part of the lemma statement is straightforward: the only chance for a process i that has not yet finished its IFS+ routine to enter the critical section in round t is that it hears the acknowledging(i) message in round $t-1$ from a process j (cf. rule (R4)). The same message is heard by all the processes except j, and thus they resign. By the algorithm definition, each process may transmit only every second round, and hence j will be silent in round t, and hears the critical message from process i. Note that if a new process starts its IFS+ routine in any round of the critical section it first listens (because of its silent phases), hears a critical message and resigns immediately. Hence, the mutual exclusion property is satisfied. □

The lemma above implies that if there is a successful transmission of an acknowledging message, then the critical section starts right after this transmission and the mutual exclusion property holds. A similar situation occurs after a successful transmission of a killing message as stated by the following lemma.

Lemma 3 *If a process successfully transmits a killing message, then it enters the critical section within the next k rounds and the mutual exclusion property holds.*

Proof After a successful transmission of a killing message by process j in round t, all remaining processes (if any) resign by rule (R1). Process j is in the last phase of its IFS+ routine and therefore enters the critical section in round $t+k$ at the latest. Note that any process activated after round t is still silent in the first round of the critical section, hears a critical message then and resigns immediately. Thus, the mutual exclusion property holds. □

Note that the cases of Lemma 2 and Lemma 3 do not cover all the possibilities of entering the critical section as the processes might start their critical section by rule (T2).

3.1.4 Bounding the makespan

In this section, we show that the makespan of the algorithm IFS+ is $O(\log n \cdot \log(1/\varepsilon))$. We note that this bound holds even in the worst-case, i.e., not only in expectation. The mutual exclusion property is not necessarily guaranteed; however, in the subsequent sections, we show that it holds with probability at least $1-\varepsilon$.

Lemma 4 *The makespan of the IFS+ algorithm is $O(\log n \cdot \log(1/\varepsilon))$.*

Proof Consider any process j that is active in round τ_{start}. Recall that the duration of its IFS+ routine is $O(k \cdot \log n)$. Let $t_{\text{end}} = \tau_{\text{start}} + O(k \cdot \log n)$ denote the last round of its IFS+ routine.

There are two possible cases. If j does not resign before t_{end}, then it enters the critical section by round $t_{\text{end}}+1$ at the latest. Otherwise, it resigns in round $t \le t_{\text{end}}$ either because of rule (R5) (it hears an announcing message) or because of rule (R1) (it hears a killing message). In the former case, the critical section starts in round t by Lemma 2; in the latter case the critical section starts in round $t+k$ at the latest by Lemma 3. Hence, in either case the makespan is $O(k \cdot \log n + k) = O(\log n \cdot \log(1/\varepsilon))$. □

3.1.5 The crucial round

The central result of this section is to show that for any starting rounds there exists a round τ_c, called *crucial round*, such that the total transmission probability (defined as the sum of transmission probabilities of all processes) in this round has just exceeded $1/4$. This guarantees that all the processes active at τ_c will not resign because of the end of their IFS+ routines for the next $2k$ rounds. In the next subsection, we show that this allows us to isolate a set of processes which are active in round τ_c and focus our analysis only on them. In particular, we show that with probability at least $1-\varepsilon$ one of them enters the critical section (preserving the mutual exclusion property) before round $\tau_c + 3k$.

To this end, we first show that the pace of the changes in the sum of transmission probabilities of active processes is restricted. For a subset of active processes B, let $P_B(t) = \sum_{i \in B} q_i(t)$. If we omit subscript B and write $P(t)$, then the sum above ranges over all active processes; we call $P(t)$ the *total transmission probability*. By the definition of the IFS+ routine, $P(\tau_{\text{start}}) = 0$ and P decreases only when some process resigns.

Lemma 5 *For any two election rounds $t \le t'$, such that $t' \le t + k$, it holds that $P(t') < 2 \cdot P(t) + 1/8$.*

Proof Let A be the set of processes that are active at time t. Note that any process that starts its IFS+ routine in a round from $[t+1, t']$ will have the transmission probability equal to 0 within this whole interval, so without loss of generality we may assume that there are no such processes.

Consider a single process $i \in A$. If i is not active in round t', then it does not contribute to $P(t')$. Otherwise, it remains active throughout the whole interval $[t, t']$ and, by the definition of IFS+, its transmission probability does not decrease there. If $q_i(t) > 0$, then $q_i(t') \le 2 \cdot q_i(t)$. If $q_i(t) = 0$, then at time t' process i is either still in the silent phase, or it is in its first non-silent phase with transmission probability $q_i(t') = 1/2^{f_i}$. Therefore, summing over all active processes, we obtain

$$P(t') \leq 2 \cdot P(t) + \sum_{i=0}^{n-1} \frac{1}{2^{f_i}},$$

where the latter summand can be bounded by

$$\sum_{i=0}^{n-1} \frac{1}{2^{f_i}} < \sum_{i=0}^{\infty} \frac{1}{2^{f_i}} = \sum_{i=0}^{\infty} \frac{1}{2^{\lceil \log(2y \cdot (i+1)^2) \rceil + 3}} \leq \sum_{i=1}^{\infty} \frac{1}{y} \cdot \frac{1}{(i+1)^2} \cdot \frac{1}{8} < \frac{1}{8}.$$

Hence, $P(t') < 2 \cdot P(t) + 1/8$. □

Lemma 6 (Crucial round) *For any adversarial choice of starting rounds t_i, there exists a round τ_c (being a deterministic function of values of t_i), such that for any execution of the IFS+ algorithm:*

1. *Either there is a process that enters the critical section (with mutex) in a round $t \in [\tau_{\text{start}} + 1, \tau_c]$,*
2. *Or τ_c is an election round, $P(\tau_c - 1) < 1/4$, and $P(\tau_c) \geq 1/4$.*

Proof First, we assume that no process ever transmits a message, and—under this assumption—we compute the value of τ_c. To this end, we take a closer look at how $P(t)$ changes as a function of time. Recall that $P(\tau_{\text{start}}) = 0$. Moreover, as we assumed no transmission, P can decrease only because some process ends its IFS+ routine. Let τ_c be the first round in which the total transmission probability reaches $1/4$, i.e., $P(\tau_c - 1) < 1/4$ and $P(\tau_c) \geq 1/4$. Note also that $\tau_c - 1$ occurs before any process starts the last phase of its IFS+ routine. (The transmission probability of such process would be $1/2$ then, which would contradict the choice of τ_c.)

Now, we take any execution of the IFS+ algorithm, and show that such choice of τ_c satisfies the conditions of the lemma. Note that in the actual execution some processes may resign before round τ_c. We consider two cases.

1. There is a successful acknowledging transmission in round $t \in [\tau_{\text{start}}, \tau_c - 1]$. By Lemma 2, in round $t+1 \leq \tau_c$ the critical section starts (with mutex).
2. There is no successful acknowledging transmission in rounds $[\tau_{\text{start}}, \tau_c - 1]$. Recall that no process can be in the last phase of its IFS+ routine in this interval, and hence there was no killing message in these rounds, either. Therefore, the only successful messages in these rounds were announcing ones, which implies that no process resigned. Thus—in terms of probability transmissions—this case is identical to the no-transmission setting we considered while defining τ_c. In particular, τ_c is an election round and it holds that $P(\tau_c - 1) < 1/4$ and $P(\tau_c) \geq 1/4$.

Note that the two cases considered above correspond to the first and the second part of the lemma statement, respectively. □

3.1.6 Successful transmissions

In the next section, we will need to ensure that within given k rounds there is a high probability of a successful transmission. Specifically, we will show the following lemma.

Lemma 7 *Fix any election round τ and assume that $P(\tau - 1) \leq 5/8$, $P(\tau) \geq 1/4$ and $q_i(\tau - 1) < 1/2$ for any active process i. Then, the probability that there is no successful transmission in the interval $[\tau, \tau + k - 1]$ is at most $\varepsilon/2$.*

Sometimes we will need a more general bound, i.e., we will require that the successful transmission is performed by a process from B, where B is some fixed subset of active processes. In this case, however, we have to clarify what happens if there is a successful transmission of a process outside of B, as such transmission may potentially terminate the entry section. In the lemma below, we assume that the effects of such transmission are ignored (e.g., the processes ignore the reception of a killing message from a process outside of B). Note that Lemma 8 immediately implies Lemma 7 when we take B as the set of all processes active in round τ. (The processes outside B are processes that become active in the interval $[\tau + 1, \tau + k - 1]$; those processes are silent in this interval by the definition of IFS+).

Lemma 8 *Fix any election round τ and assume that $P(\tau - 1) \leq 5/8$ and $q_i(\tau - 1) < 1/2$ for any process i. Fix any subset B of processes active in round τ, such that $P_B(\tau) \geq 1/4$. Assume that processes from B ignore the successful transmission performed by processes not in B. Then, the probability that there is no successful transmission performed by a process from B in the interval $[\tau, \tau + k - 1]$ is at most $\varepsilon/2$.*

Proof Divide all rounds from the interval $[\tau, \tau + k - 1]$ into odd-numbered and even-numbered ones, interleaved. Note that any process transmits either only in even or only in odd rounds: this partitions set B into two parts: B' transmitting in odd rounds and B'' transmitting in even rounds. Without loss of generality, we may assume that $P_{B'}(\tau) \geq 1/8$.

As $q_i(\tau - 1) < 1/2$, no process ends its IFS+ routine before round $\tau + k - 1$. We focus on any odd round $t \in [\tau, \tau + k - 1]$ and assume that there was no successful transmission by a process from B until round $t - 1$. What happens in round t? Any process $j \in B'$ transmits with probability $q_j(t)$ and an alive process $j \notin B$ either transmits with probability $q_j(t)$ or does not transmit at all (because by the definition of IFS+ any process may transmit only every second round). Let C be the set of the processes not in B that may transmit in round t. Then, the probability of a successful transmission by a process from B in round t is equal to

$$p_t = \sum_{j \in B'} q_j(t) \prod_{i \in B' \cup C \text{ and } i \neq j} (1 - q_i(t)).$$

We first bound the product occurring in this equation. Let A be the set of all processes active in round t. As for any process i it holds that $q_i(t) \leq 1/2$, we have $1 - q_i(t) \geq (1/4)^{q_i(t)}$. Hence,

$$\prod_{i \in B' \cup C \text{ and } i \neq j} (1 - q_i(t)) \geq \prod_{i \in A}(1 - q_i(t)) \geq \prod_{i \in A}(1/4)^{q_i(t)} = (1/4)^{\sum_{i \in A} q_i(t)} = (1/4)^{P(t)}.$$

Moreover, by Lemma 5, it holds that $P(t) < 2 \cdot P(\tau - 1) + 1/8 \leq 11/8$. As there was no successful transmissions by a process from B in rounds $[\tau, t-1]$ and the successful transmissions by processes not in B were ignored, no process from B resigned. Therefore, $P_{B'}(t) \geq P_{B'}(\tau)$, and hence

$$p_t \geq \sum_{j \in B'} q_j(t) \cdot \left(\frac{1}{4}\right)^{11/8} > \frac{1}{8} \cdot \frac{1}{7} = \frac{1}{56}.$$

Therefore, the probability of no successful transmission (by a process from B) in odd rounds is at most $(55/56)^{k/2} \leq \varepsilon/2$. The probability of no successful transmission (by a process from B) within whole interval $[\tau, \tau + k - 1]$ is at most that, which implies the lemma. □

3.1.7 Bounding the failure probability

We consider the crucial round τ_c, whose existence is guaranteed by Lemma 6. We call the interval $[\tau_c + 1, \tau_c + 3k]$ *crucial interval*, and we show that with probability at least $1 - \varepsilon$ some process enters the critical section in the crucial interval. In this section, we make two assumptions in all our lemmas and their proofs:

1. τ_c is an election round;
2. No process starting its entry section during the crucial interval ever transmits in that interval.

Indeed, Lemma 6 states that if the former assumption is not true, then some process already entered the critical section (with mutex) before or in round τ_c. To justify the latter assumption, observe that even if a process becomes active in a round $t > \tau_c$, it will be silent until round $t+3k-1 \geq \tau_c+3k$ (inclusively). Hence, when a process in the critical section transmits its first critical message, all these silent processes resign.

A process j is called *the leader* if it is announcing or killing and all other processes (if any) are acknowledging(j).

Lemma 9 *With probability at least* $1 - \varepsilon/2$:

1. *Either some process enters the critical section (with mutex) until round* $\tau_c + k$ *(inclusively);*
2. *Or* $\tau_c + k$ *is an election round, there exists a leader in this round, and no process resigned in rounds from* $[\tau_c, \tau_c + k - 1]$.

Proof Let A be the set of processes active in round τ_c. By Lemma 6, $P(\tau_c - 1) < 1/4$ and $P(\tau_c) \geq 1/4$, and thus for any process i it holds that $q_i(\tau_c - 1) \leq 1/8$. Hence, IFS+ routine of any process from A lasts at least until round $\tau_c + 2k - 1$ and no active process is killing in rounds $[\tau_c, \tau_c + k - 1]$. By Lemma 7, with probability at least $1 - \varepsilon/2$, there is a successful transmission within these rounds (performed by an announcing or acknowledging process). It remains to show that the existence of a successful transmission implies the lemma statement. We consider two cases.

1. If there is any successful transmission of an acknowledging message in rounds $[\tau_c, \tau_c + k - 1]$, then by Lemma 2, the critical section starts (with mutex) at the subsequent round.
2. Otherwise, there is no transmission of an acknowledging message. In this case, there are only transmissions of announcing messages in rounds from $[\tau_c, \tau_c + k - 1]$, and hence no process resigns in these rounds (as no process is killing). Let $t \in [\tau_c, \tau_c + k - 1]$ be the last round with a successful transmission. Then in round $t+1$, all processes except j are in state acknowledging(j) and process j becomes the leader. As there are no subsequent successful transmissions until round $\tau_c + k - 1$ (inclusively) and no new process that becomes active until round $\tau_c + k$ (inclusively) transmits, the situation remains unchanged until round $\tau_c + k$. □

Lemma 10 *Assume that: no process resigned in rounds* $[\tau_c, \tau_c+k-1]$, τ_c+k *is an election round, and there is a leader* j *in this round. Then, with probability at least* $1 - \varepsilon/2$*, some process enters the critical section (with mutex) in a round from* $[\tau_c + k + 1, \tau_c + 3k - 1]$.

Proof By the lemma assumptions, in round τ_c+k there exists one process j in announcing or killing state and all other processes (if any) are acknowledging(j). As $P(\tau_c - 1) < 1/4$ (cf. Lemma 6), Lemma 5 implies that $P(\tau_c + k - 1) < 2 \cdot 1/4 + 1/8 = 5/8$. As $P(\tau_c) \geq 1/4$ and no process resigns in rounds from $[\tau_c, \tau_c + k - 1]$, it holds that $P(\tau_c + k) \geq 1/2$.

We now take a closer look at what may happen in the interval $[\tau_c + k, \tau_c + 2k - 1]$. As $q_j(\tau_c - 1) \leq 1/8$, the IFS+ routine of j lasts at least until round $\tau_c + 2k - 1$ and j is either in the killing state in all rounds from the interval $[\tau_c + k, \tau_c + 2k - 1]$ or it is in the announcing state in round $\tau_c + k$ and potentially switches to the killing state later. There can be three types of successful transmissions in the interval $[\tau_c + k, \tau_c + 2k - 1]$.

1. If process j transmits successfully while in the announcing state, then this transmission has no effect at all (as all other processes are already acknowledging(j)).
2. If process j transmits successfully while in the killing state, the critical section (with mutex) starts within the next k rounds (i.e., in round $\tau_c + 3k - 1$ at the latest) by Lemma 3.
3. If there is a successful transmission of an acknowledging (j) message, then the critical section (with mutex) starts in the next round by Lemma 2.

Now, we show that there will be a successful transmission of the second or third type with probability at least $1 - \varepsilon/2$. To this end, we consider two cases.

1. $q_j(\tau_c + k) > 1/4$. Since transmission probabilities are powers of $1/2$, it actually holds that $q_j(\tau_c + k) = 1/2$, i.e., process j is killing already in round $\tau_c + k$. In this case, one successful transmission is sufficient, and the claim follows immediately by Lemma 7.
2. $q_j(\tau_c + k) \leq 1/4$. As the total transmission probability in round $\tau_c + k$ is at least $1/2$, the total probability of processes acknowledging(j) is at least $1/4$.
   Consider any *real* execution of the protocol in rounds $[\tau_c + k, \tau_c + 2k - 1]$ and the corresponding *virtual* execution obtained by replacing any killing messages of process j by announcing messages. In the virtual execution, all transmissions of process j can be ignored (as other processes are in acknowledging(j) state already), and hence, by Lemma 8, there is a successful transmission of the message acknowledging(j) in this interval, with probability at least $1-\varepsilon/2$. In such case, the mutual exclusion condition holds for the considered virtual execution (in rounds $[\tau_c+k, \tau_c+2k-1]$). Note that it holds then also in the corresponding real execution: if the first killing message occurs before the first acknowledging message in the real execution, the mutual exclusion is guaranteed by Lemma 3, otherwise the real execution is the same as the virtual one (in which, as we just showed, the mutual exclusion condition holds as well). □

Theorem 4 *The algorithm IFS+ guarantees ε-mutual-exclusion with makespan $O(\log n \cdot \log(1/\varepsilon))$.*

Proof By Lemma 4, the makespan of the algorithm IFS+ is $O(\log n \cdot \log(1/\varepsilon))$. It remains to show that the the mutual exclusion property holds with probability at least $1 - \varepsilon$.

To this end, we fix any adversarial choice of starting rounds t_i; this defines starting round τ_{start}. Let $\tau_c = \tau_{\text{start}} + O(k \cdot \log n)$ be the crucial round guaranteed by Lemma 6, in the following sense: either the critical section starts already (with mutex) in the interval $[\tau_{\text{start}} + 1, \tau_c]$ or τ_c is an election round. In the latter case, Lemma 9 guarantees either start of the critical section (with mutex) in rounds $[\tau_c + 1, \tau_c + k]$ or the precondition for Lemma 10, with error probability at most $\varepsilon/2$. On the other hand, Lemma 10 guarantees the start of the critical section (with mutex) in rounds $[\tau_c + k + 1, \tau_c + 3k - 1]$, with error probability at most $\varepsilon/2$. Thus, the total probability that the mutual exclusion property does not hold is at most ε. □

3.2 Collision detection available

In this section, we show two algorithms working in a scenario with collision detection. First, we show algorithm StaticQSEB- Emulation that solves the *static case* of the ε-mutual-exclusion problem, i.e., the case where there is a subset of processes which start their entry sections in round 1 and no process is activated later. Then, we show algorithm DynamicQSEB- Emulation which solves ε-mutual-exclusion problem in (asymptotically) the same makespan. In what follows, we assume that whenever a process does not transmit, it listens.

3.2.1 Solving the static case

Protocol StaticQSEB- Emulation starts with EnterIfSingle subroutine to detect (with probability at least $1 - \varepsilon$) if there is only one active process. In such case, this process enters the critical section. Otherwise, Willard's QSEB algorithm [13] is simulated in order to choose the only process for entering the critical section.

The subroutine EnterIfSingle assumes that there is a set of processes that start this procedure simultaneously, and thus that rounds are numbered, starting from 1. It consists of $2 \cdot \log(1/\varepsilon)$ rounds. In each odd round, each active process tosses a symmetric coin (i.e., with probability $1/2$ of success) to choose whether it transmits in the current round and listens in the next round, or vice versa. If the process never hears anything, it enters the critical section at the end of the procedure.

Lemma 11 *Assume ℓ processes execute the procedure* EnterIfSingle. *If $\ell = 1$, then the only process enters the critical section. If $\ell \geq 2$, then with probability $1 - \varepsilon$, no process enters the critical section.*

Proof The first claim holds trivially. For showing the second one, we fix an odd-even pair of rounds. Let E denote the event that there is a process that hears neither signal nor collision in this pair of rounds. For this to happen all processes running CheckIfSingle have to transmit in the odd round or all have to transmit in the even round. Thus, $\Pr[E] = 2 \cdot 1/2^{\ell} = 1/2^{\ell-1} \leq 1/2$. Since the transmissions in different pairs of rounds are independent, the probability that there exists a process that does not hear anything during the whole algorithm, and thus enters the critical section, is at most $(1/2)^{\log(1/\varepsilon)} = \varepsilon$. □

If no process enters the critical section at the end of EnterIfSingle, we simulate Willard's algorithm QSEB [13].

Although it can be treated as a black box, we give a brief overview below. This algorithm is a selection protocol that runs on a multiple access channel in settings with collision detection and unknown number of processes (even more: without any upper bound on n). The aim is to select a single process broadcasting in a single round. It is assumed that each process can broadcast and listen in the same round (which is not the case in our model). The expected time of this algorithm is $\log\log n + o(\log\log n)$ and can be regarded as an extension of superexponential binary search (SEBS) (also in [13]). In both protocols, the first phase is devoted to finding (with high probability) c, such that $c \leq \ell \leq 2c$ and the actual number of processes is $n = 2^{\ell}$. The first phase takes at most $\log\log n$ rounds. In each round, each process broadcasts independently with fixed probability. Collision is an evidence that the probability is too high. Similarly, silence suggests that the probability is too low. Using these observations one can estimate ℓ in $\log \ell$ rounds. The second phase is based on consecutive broadcasting, with probability 2^{-c} or 2^{-2c}, until a single process is broadcasting. This procedure leads to selecting a single station in $O(1)$ rounds (in expectation) provided that c was properly found. To avoid the necessity of using a bound on n, QSEB uses an algorithm for unbounded searching from [21].

In our settings, a simulation of the original Willard's algorithm is required, as the original algorithm of [13] assumed that each process can simultaneously transmit and listen in each round. The QSEB algorithm is simulated in the following way.

As we consider a static setting, it is possible to distinguish odd and even rounds. In odd rounds processes run the simulated protocol. Non-transmitting processes listen on the channel. If in an odd round $2k + 1$ a process hears a successful transmission, it transmits a message in the subsequent even round $2k + 2$. Each process that transmitted in round $2k + 1$, listens in round $2k + 2$. If a process transmitted in round $2k + 1$ and hears a transmission or collision in round $2k + 2$, it enters the critical section in round $2k + 3$.

First, we show what happens in two rounds of algorithm StaticQSEB- Emulation when simulating one round of Willard's algorithm QSEB.

Lemma 12 *In the static case, if there are at least two active processes, algorithm* StaticQSEB- Emulation *simulates one round taken in the model in which a process may simultaneously transmit and listen by two rounds in the model in which a process is allowed to either transmit or listen, provided that collision detection is available.*

Proof Since there are at least 2 processes, if exactly one process transmits successfully in round $2k + 1$ (and thus all other stations hear it), then this process hears a signal or a collision in round $2k + 2$. If this process was not the only transmitter in round $2k + 1$, then it hears silence in round $2k + 2$. This allows any process transmitting in round $2k + 1$ to recognize (in round $2k + 2$) whether it was a single (and thus) successful transmitter in that round or not (i.e., there was a collision). All other processes recognize it, and receive a successful transmission if there is one, in round $2k + 1$. □

Theorem 5 *In the scenario with collision detection, algorithm* StaticQSEB- Emulation *solves the static ε-mutual-exclusion problem with expected makespan $O(\log\log n + \log(1/\varepsilon))$.*

Proof If there is only one process starting its entry section, it enters the critical section at the end of EnterIfSingle, which takes $O(\log(1/\varepsilon))$ rounds. If there are at least two processes, with probability $1 - \varepsilon$, none of them enters the critical section at the end of this procedure and they all simultaneously start the simulation of Willard's algorithm QSEB. By the property of Willard's algorithm QSEB [13] and by Lemma 12, in expectation there is a successful transmission in $O(\log\log n)$ rounds. □

Note that this result matches logarithmic lower bound expressed in Theorem 2 as well as Theorem 2 from [3] for $\epsilon = 1/n^c$.

3.2.2 The algorithm for the dynamic case

It remains to show that we can use algorithm StaticQSEB- Emulation to solve the general (i.e., dynamic) version of the ε-mutual-exclusion problem. The idea behind algorithm DynamicQSEB- Emulation is to synchronize processes at the beginning, and then to transmit a "busy" signal in every second round. New processes starting their entry section note this signal within their first two rounds and will not be competing for the critical section, until an exit section releases the shared channel.

Algorithm DynamicQSEB- Emulation for a particular process p is constructed as follows. Upon starting the entry section, process p listens for two rounds. If it hears a transmission or a collision in either of these rounds, it resigns. Otherwise, it starts counting in the following rounds. In odd rounds (starting from round 3) it always transmits a message. In even rounds, starting from round 4, it emulates consecutive rounds of the entry section of algorithm StaticQSEB- Emulation. If at some point of the emulation, algorithm

StaticQSEB- Emulation decides that process p must enter the critical section, then it does so in the main execution of algorithm DynamicQSEB- Emulation.

Theorem 6 *Assume the model with collision detection. Algorithm* DynamicQSEB- Emulation *solves the (dynamic) ε-mutual-exclusion problem with expected makespan $O(\log\log n + \log(1/\varepsilon))$.*

Proof Let t be a round in the execution in which there is at least one process in the entry section, no process is in the exit or critical section, and such that there was no process in the entry section in the previous round $t-1$. Let P denote the set of processes that are in the entry section in round t. In rounds t and $t+1$ they all listen and hear silence. Every second round, starting from round $t+2$, all processes from P transmit. We call these rounds *signal rounds*. If a process is activated in round $t+1$ or later, one of its initial two rounds is a signal round. Hence, it hears a transmission or a collision and resigns. Consequently, in rounds $t+3, t+5, t+7, \dots$ only processes from P emulate their entry sections from algorithm StaticQSEB- Emulation, and they all start execution of algorithm StaticQSEB- Emulation in round $t+3$. By Theorem 5, with probability at least $1-\varepsilon$, exactly one process enters the critical section by round $t+1+2T$, where T is the random variable denoting the makespan of algorithm StaticQSEB- Emulation. Hence the expected makespan of DynamicQSEB- Emulation is $2+2T = O(\log\log n + \log(1/\varepsilon))$. □

4 Fairness

The algorithms shown in [3] and in Sect. 3 do not consider the no-lockout property, i.e., it may happen that a process never gets out of its entry section, as other processes exchange access to the critical section among themselves. We show how to modify algorithms satisfying the no-deadlock property (in particular, the algorithms from [3] and those of Sect. 3), so that the no-lockout property is fulfilled. Moreover, our transformation allows to express the (expected) makespan of obtained fair protocols in terms of the (expected) makespan of the original weaker protocols.

The rough idea of the transformation is as follows: Each process maintains an additional local counter of losses, denoting how many times it competed for the critical section with other processes and lost. When a process enters its exit section, it becomes a *guard*: it helps processes currently being in the entry section to choose one of them with the highest loss counter. This ensures that each process that initiated entry section eventually gains access to the critical section, i.e., the no-lockout property.

We start by constructing two routines that, given a distinguished guarding process and some set of active processes, choose an active process with the highest counter. We assume that each active process i keeps a loss counter $k_i \in \{0, \dots, n-1\}$ (we later justify the assumption $k_i < n$). The first routine assumes the KN scenario, the second one—the CD scenario; they are called ChooseHighest- KN and ChooseHighest- CD, respectively. Later, we show a generic algorithm that uses either of these routines as a black box and guarantees the no-lockout property. Except executing the black box, the algorithm will not require any capabilities of the channel (such as KN, CD or GC).

4.1 Choosing the highest counter in KN scenario

We start with a description of ChooseHighest- KN routine. Recall, that in the KN scenario, the number of processes, n, is known to each process. Let $\ell = \lceil \log n \rceil$, i.e., the number of bits needed to encode any ID. Let B_i be the binary representation of $k_i \cdot 2^\ell + i$ padded with leading zeros so that its length is 2ℓ. That is, B_i is the concatenation of the bit representation of k_i and the process ID.

ChooseHighest- KN takes 4ℓ rounds numbered from 1 to 4ℓ. The distinguished *guarding* process transmits a *hello* message in each odd round. In each even round but the last one, it transmits message *guarding_in_progress* and in the last even round 4ℓ, it transmits a message *guarding_end*. The remaining processes may be either active at the beginning of ChooseHighest- KN or inactive; in the latter case they remain silent for the whole routine. Each active process i listens in even rounds, while in an odd round $2s-1$ its action depends on the s-th highest bit from B_i. If this bit is 1, process i transmits a message. Otherwise this bit is 0, and process i listens on the channel. If it does not hear a successful *hello* message (which means that there was some other process transmitting at that round), it becomes inactive till the end of the routine.

Lemma 13 ChooseHighest- KN *takes $O(\log n)$ rounds. In all even rounds* guarding *types of messages are successfully transmitted. If the subset of processes active at the beginning of* ChooseHighest- KN *is nonempty, then at the end of routine* ChooseHighest- KN *exactly one active process remains and it has maximal k_i among all processes that were active at the beginning of the routine.*

Proof The first two properties follow trivially. For showing the third one, let i be the process whose B_i is lexicographically last among other B_j strings. Clearly, $k_i \geq k_j$ for any other process j. Choose any other process j. By the choice of i, there is a (possibly empty) prefix of B_j that is equal to the prefix of B_i, and then they differ on some bit whose value is 1 in B_i and 0 in B_j. Hence, at the odd round corresponding to this bit, process i transmits, while process j listens and becomes inactive. For the same reason, process i never becomes inactive during ChooseHighest- KN. □

4.2 Choosing the highest counter in CD scenario

Note that in the routine ChooseHighest- KN, it is not necessary that the processes know the exact value of n. They only have to know a common integer value ℓ that is large enough, so that ℓ bits are sufficient to encode any ID of an active process and the value of its counter.

This suggest the following approach for the CD scenario: first run a ComputeL routine (described below), at the end of which all active processes and the guarding process share the common value of ℓ satisfying the property above and execute ChooseHighest- KN immediately afterwards, using this value of ℓ. The concatenation of these routines is denoted ChooseHighest- CD.

Routine ComputeL: Non-active processes are silent throughout the whole ComputeL routine. An active process i picks $\ell_i = \max\{\lceil \log i \rceil, \lceil \log k_i \rceil\}$. Again, rounds are divided into odd and even ones. An active process i transmits a *hello* message in the first ℓ_i odd rounds, i.e., in rounds $1, 3, \ldots, 2\ell_i - 1$, it is silent in the remaining odd rounds and listens in all even rounds. The guarding process listens in all odd rounds. If it hears a *hello* message or a collision (i.e., at least one active process tranmits), then in the subsequent even round it transmits a *guarding_counting_in_progress* message. Otherwise (no active process transmits) it transmits *guarding_counting_ends* message, which is heard by all other processes and ComputeL routine terminates. All active processes choose ℓ to be the total number of odd rounds that elapsed in this part of the routine minus 1, that is, ℓ is the maximum of ℓ_i values among all active processes.

As the number of rounds ComputeL requires is even and bounded by $O(\log n)$, we immediately obtain a counterpart of Lemma 14 for the CD scenario.

Lemma 14 ChooseHighest- CD *takes* $O(\log n)$ *rounds. In all even rounds* guarding *types of messages are successfully transmitted. If the subset of processes active at the beginning of* ChooseHighest- CD *is nonempty, then at the end of routine* ChooseHighest- CD *exactly one active process remains and it has maximal* k_i *among all processes that were active at the beginning of the routine.*

4.3 Ensuring the no-lockout property

Now, we describe a transformation of a mutual exclusion algorithm A with empty exit sections into an algorithm A' that additionally satisfies the no-lockout property. Our transformation will use ChooseHighest- KN routine for KN scenario and ChooseHighest- CD routine for CD scenario. From now on, we denote this routine simply by ChooseHighest. We describe how to modify particular sections of process i. Each process i will maintain a counter k_i that is initially set to zero.

– *Critical section.* If the critical section of a process is empty in algorithm A, it now lasts one round, i.e., in A' process i sends at least one critical message.
– *Exit section.* The exit section of process i consists of $O(\log n)$ rounds. In the first round, it transmits *guarding_start* message and later it executes the ChooseHighest routine acting as guarding process. Note that this routine always ends with the *guarding_end* message.
– *Entry section.* We denote the original entry section routine from algorithm A by E_i. Process i first listens for 2 rounds. If it does not hear any *guarding* type of message (*guarding_start*, *guarding_in_progress*, *guarding_end*, *guarding_counting_in_progress* or *guarding_counting_ends*) in any of these two rounds, it executes E_i. Otherwise, it waits till it hears message *guarding_end* and it starts E_i in the subsequent round. The routine E_i is however stopped when another process enters its critical section. This can be trivially achieved by listening on each round when process i is not transmitting: as A guarantees the mutual exclusion property, the process is then required to listen whenever other processes send critical messages. After E_i is stopped, process i sets $k_i = 0$ and repeats the following scheme until it enters the critical section: It waits until the process currently in its critical section initiates its exit section (by transmitting the *guarding_start* message). In the following round, process i initiates ChooseHighest routine as an active process. If at the end of ChooseHighest routine it is still active, it enters the critical section, otherwise it increments k_i and repeats the scheme.

Lemma 15 *Assume that the number of processes is known or the collision detection is available, and an algorithm A solves the mutual exclusion problem (or the* ϵ*-mutual exclusion problem) with (expected) makespan* T*. Moreover, assume that the exit section of A is empty. Then the transformation described above yields an algorithm* A' *that also guarantees the no-lockout property and has (expected) makespan* $T + O(\log n)$*.*

Proof First, observe that messages of *guarding* type are always successfully transmitted and heard by all active processes. Thus, in the following, for the purpose of analysis only, we may remove all initial waiting periods from the execution. This means that (i) the entry section of process j starts already with E_j, (ii) it never starts during an exit section of another process (it may start right after the exit section ends, though), and (iii) the actual execution of an entry section might be longer at most by the length of the exit section, i.e., by $O(\log n)$ rounds.

We define *non-CE phase* as the maximum sequence of contiguous rounds during which no process is in its critical or exit section. We now show that A' satisfies the mutual exclusion property and the following invariants hold:

1. A critical section of the executed process j is followed by an exit section of j.
2. During exit section of some process j, the only transmitting processes are those participating in ChooseHighest routine (denote their set by S), with j acting as guarding process. If S is nonempty, then the exit section of j is followed immediately by a critical section. Otherwise it is followed by a non-CE phase.
3. If a process j is in its entry section within a *non-CE phase*, it is executing E_j routine.

The proof follows by a simple induction. The algorithm starts in a non-CE phase. Within any non-CE phase, the processes in their entry sections execute their E_i routines, and finally one of them, say j, enters the critical section with the mutual exclusion property (ϵ-mutual-exclusion property) guaranteed by algorithm A. In the critical section of j, all other processes that are in their entry sections are simply waiting for the *guarding_start* message, and clearly such a section is followed by an exit section of j. Let S be the set of processes that are in their entry sections during exit section of j. By our assumption, their entry sections must have started before the exit section of j, and hence they are now executing ChooseHighest routine. Thus, if S is nonempty, then after the exit section of j, exactly one process from S (one with the maximal counter k_i) starts its critical section. If S is empty, then the exit section is followed by the non-CE phase.

For bounding the makespan, we observe that if a critical section started after an exit section, then the makespan is at most the length of this exit section, i.e., $O(\log n)$. Otherwise (a critical section starts after a non-CE phase), the processes execute their E_j routines, and hence the time between any of them started its entry section and the critical section is at most T by the property of algorithm A. Recall that we are considering a modified execution (without initial waiting periods), and thus the actual makespan is $T + O(\log n)$. (In the actual execution, the processes may have started their entry sections already during the exit section preceding the non-CE phase in question.)

For showing the no-lockout property, observe that if a non-CE phase starts, then in the exit section preceding this non-CE phase, there was already no process in its entry section. Thus, we only have to show that the no-lockout property is satisfied in a sequence of interleaved critical and exit sections. Note that a process may lose the competition in ChooseHighest routine at most $n - 1$ times before it is guaranteed to have the highest k_i counter among the processes in their entry sections. Even if other processes execute new entry sections after they finish their critical sections, they do so with their counters reset to 0. Thus, each process eventually enters the critical section. □

By combining Lemma 15 with the results from Section 3 and with the existing no-deadlock deterministic algorithms of [3], we obtain the following two results.

Corollary 1 *There exists a randomized algorithm with the no-lockout property with expected makespan $O(\log n + \log(1/\varepsilon))$ solving the ε-mutual-exclusion problem in the model in which collision detection is available, and a randomized algorithm with the no-lockout property with makespan $O(\log n \cdot \log(1/\varepsilon))$ in the model with known number of processes.*

Corollary 2 *There exists a deterministic algorithm with the no-lockout property with makespan $O(\log n)$ solving the mutual exclusion problem in a model in which collision detection is available and a deterministic algorithm with the no-lockout property with makespan $O(n \log^2 n)$ in a model with known number of processes.*

5 Conclusions and open problems

In this paper, we presented several results about mutual exclusion problem on a multiple access channel. In particular, we relaxed the classical notion of the mutual exclusion problem to the ε-mutual-exclusion problem and showed that the relaxed condition can be guaranteed with at least exponential speed-up. Finally, we showed how to achieve an additional property of no-lockout within an additional logarithmic cost.

Finding relationship between contention resolution problem and the mutual exclusion problem with the no-lockout property seems to be an interesting but challenging task. It would be also interesting to investigate if it possible to transfer the lower bounds proved for related models and problems, such as wake-up or leader election on a single-hop radio network, to the ε-mutual-exclusion problem.

In this work, we did not study energy consumption and fault-tolerance of mutual exclusion protocols, which are potentially interesting and important open problems. Another perspective aspect is to study the problem of k-set-mutual exclusion, in which there are k available channels.

References

1. Attiya, H., Welch, J.: Distributed Computing: Fundamentals, Simulations and Advanced Topics. Wiley, New York (2004)
2. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Publishers Inc, Burlington (1996)
3. Czyzowicz, J., Gasieniec, L., Kowalski, D.R., Pelc, A.: Consensus and mutual exclusion in a multiple access channel. IEEE Trans. Parallel Distrib. Syst. **22**(7), 1092–1104 (2011)
4. Bar-Yehuda, R., Goldreich, O., Itai, A.: On the time-complexity of broadcast in multi-hop radio networks: an exponential gap between determinism and randomization. J. Comput. Syst. Sci. **45**(1), 104–126 (1992)
5. Capetanakis, J.: Tree algorithms for packet broadcast channels. IEEE Trans. Inf. Theory **25**(5), 505–515 (1979)
6. Clementi, A.E.F., Monti, A., Silvestri, R.: Selective families, superimposed codes, and broadcasting on unknown radio networks. In: Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 709-718 (2001)
7. Greenberg, A.G., Winograd, S.: A lower bound on the time needed in the worst case to resolve conflicts deterministically in multiple access channels. J. ACM **32**(3), 589–596 (1985)
8. Jurdzinski, T., Kutylowski, M., Zatopianski, J.: Efficient algorithms for leader election in radio networks. In: Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC), pp. 51-57 (2002)
9. Kowalski, D.R.: On selection problem in radio networks. In: Proceedings of the 24th ACM Symposium on Principles of Distributed Computing (PODC), pp. 158-166 (2005)
10. Kushilevitz, E., Mansour, Y.: An omega(D log (N/D)) lower bound for broadcast in radio networks. SIAM J. Comput. **27**(3), 702–712 (1998)
11. Nakano, K., Olariu, S.: Uniform leader election protocols for radio networks. IEEE Trans. Parallel Distrib. Syst. 13(5), 516–526 (2002)
12. Tsybakov, B.S., Mikhailov, V.A.: Free synchronous packet access in a broadcast channel with feedback. Problemy Peredachi Informatsii **14**(4), 32–59 (1978)
13. Willard, D.E.: Log-logarithmic selection resolution protocols in a multiple access channel. SIAM J. Comput. **15**(2), 468–477 (1986)
14. Chlebus, B.S., Gasieniec, L., Kowalski, D.R., Radzik, T.: On the wake-up problem in radio networks. In: Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP), pp. 347-359 (2005)
15. Gasieniec, L., Pelc, A., Peleg, D.: The wakeup problem in synchronous broadcast systems. SIAM J. Discrete Math. **14**(2), 207–222 (2001)
16. Jurdzinski, T., Stachowiak, G.: Probabilistic algorithms for the wakeup problem in single-hop radio networks. In: Proceedings of the 13th International Symposium on Algorithms and Computation (ISAAC), pp. 535-549 (2002)
17. Jurdzinski, T., Stachowiak, G.: The cost of synchronizing multiple-access channels. In: Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC), pp. 421-430 (2015)
18. Goldberg, L.A., Jerrum, M., Kannan, S., Paterson, M.: A bound on the capacity of backoff and acknowledgment-based protocols. SIAM J. Comput. **33**(2), 313–331 (2004)
19. Bender, M.A., Farach-Colton, M., He, S., Kuszmaul, B.C., Leiserson, C.E.: Adversarial contention resolution for simple channels. In: Proceedings of the 16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 325-332 (2005)
20. Chlebus, B.S., Kowalski, D.R., Rokicki, M.A.: Adversarial queuing on the multiple access channel. ACM Trans. Algorithms **8**(1), 5 (2012)
21. Bentley, J.L., Yao, A.C.: An almost optimal algorithm for unbounded searching. Inf. Process. Lett. **5**(3), 82–87 (1976)