Contention Resolution with Message Deadlines*

Kunal Agrawal
Washington University in St. Louis
St. Louis, MO 63130-4899, USA
kunal@wustl.edu

Michael A. Bender
Stony Brook University
Stony Brook, NY 11794-2424, USA
bender@cs.stonybrook.edu

Jeremy T. Fineman
Georgetown University
Washington, DC 20057-1232, USA
jfineman@cs.georgetown.edu

Seth Gilbert
National University of Singapore
Singapore 117417
seth.gilbert@comp.nus.edu.sg

Maxwell Young
Mississippi State University
Mississippi State, MS 39762, USA
myoung@cse.msstate.edu

ABSTRACT

In the contention-resolution problem, multiple players contend for access to a shared resource. Contention resolution is used in wireless networks, where messages must be transmitted on a shared communication channel. When two or more messages are transmitted at the same time, a collision occurs, and none of the transmissions succeed. Much of the theoretical work on contention resolution has focused on efficiently resolving collisions in order to obtain throughput guarantees.

However, in modern-day networks, not all traffic is treated equally. Instead, messages are often handled according to a notion of priority. While throughput remains an important metric, it fails to capture this increasingly-common scenario of traffic prioritization.

Motivated by this concern, we design a contention-resolution algorithm where messages have delivery deadlines. Unit-length messages dynamically arrive over time, each with a corresponding delivery deadline that demarcates a window of time wherein the message must be transmitted successfully. We consider inputs that have a feasible schedule, even if message sizes increase by a constant factor. In this setting, we provide an algorithm which guarantees that each message succeeds by its deadline with high probability in its window size.

CCS CONCEPTS

• **Theory of computation → Distributed algorithms**; • **Networks → Network algorithms**.

KEYWORDS

Contention resolution, randomized backoff, scheduling, deadlines

*This work was supported in part by NSF grants CNS-1816076, CCF-1733873, CCF-1718700, CCF-1918989, CCF-1725543, CSR-1763680, CCF-1716252, CCF-1617618, CNS-1938709, and by Singapore MOE Tier 2 grant MOE2018-T2-1-160.

SPAA '20, July 15–17, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-6935-0/20/07…$15.00
https://doi.org/10.1145/3350755.3400239

ACM Reference Format:
Kunal Agrawal, Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Maxwell Young. 2020. Contention Resolution with Message Deadlines. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '20), July 15–17, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3350755.3400239

1 INTRODUCTION

In the abstract ***contention-resolution*** problem, there are multiple players contending for exclusive (but temporary) access to a shared resource [46, 72]. Typically the shared resource is modeled as a communication channel.[1] Players arrive over time with the goal of successfully transmitting one message on the channel. The communication medium is formalized as a ***multiple access channel***. Time consists of a sequence of synchronized ***slots***. In each slot, a player may transmit a message, but the transmission ***succeeds*** only if no other players transmit during the same slot. If two or more players transmit a message in the same slot, then a ***collision*** occurs and all transmissions during that slot fail.

In this paper, we present a contention-resolution algorithm for the setting where players have deadlines by which they must deliver their messages. Delivering messages with deadlines is a natural problem that has been considered in different contexts [44, 73, 74]. For instance, real-time communication protocols such as WirelessHART [43], RT-Link [2], and Glossy [40] are used in industrial systems and must ensure that a message from the sensor is delivered within a certain time in order for it to be useful. More generally, deadlines capture a notion of priority and, in turn, address starvation and fairness, which are important to user experience [55, 58, 71].

The classic contention-resolution algorithm is randomized exponential backoff [72]. For example, IEEE 802.11 (WiFi) networks make use of binary exponential backoff to transmit packets on a wireless channel [1]. It has been known for many years that exponential backoff does not guarantee an optimal makespan (see e.g., [13, 14, 91]), and there has been extensive theoretical work on developing asymptotically-optimal algorithms [8, 16, 45, 52]. However, none of these algorithms are designed to meet real-time constraints on packet delivery.

[1] Although randomized backoff is usually thought of as applying to wired [72] or wireless networks [64, 89], it has other applications, including shared memory [10, 54], lock acquisition [81], email retransmission [19, 37], and congestion control (e.g., TCP) [57, 75].

Randomized Backoff: We briefly describe exponential backoff in order to further motivate deadlines. When a player p has a packet to send on the channel, the player keeps attempting broadcasts until it has a successful transmission. Whenever a broadcast by p fails (due to a collision), then p waits for a random amount of time, proportional to how long it has been in the system and then attempts another broadcast.

Even without deadlines, backoff algorithms have the drawback that they can enable starvation, potentially shutting out a player for a long amount of time. A newly-arrived player may get to send its message quickly, ahead of players that arrived previously but suffered through failed transmission attempts and ratcheted down their broadcast probabilities. Message deadlines can be used as a mechanism to both avoid starvation and to specify any timing constraints that the messages might have.

1.1 Model and Problem

We model contention resolution with deadlines as follows. An instance consists of a set of n ***jobs*** that arrive over time. Each job j has a release time r_j, a deadline d_j, and a unit-length ***message*** to send. (When there is ambiguity, we refer to this message as a ***data message***.) The goal of each job j is to broadcast its data message on the channel during a slot falling in the time interval $[r_j, d_j]$, which we refer to as j's ***window***. We use $w_j = d_j - r_j$ to denote the ***window size***. Job j is activated at its release time r_j, and the job may only interact with the channel during its window.

Each job has limited *a priori* knowledge. Jobs do not have distinct IDs, nor do they have access to a global clock. Thus, when a job j is activated, it does not know the value r_j. Job j only knows that the deadline will arrive w_j slots later. The only communication between jobs is via the multiple access channel. To facilitate coordination, jobs may transmit additional messages, called ***control messages***, which are distinct from ***data messages*** that they are required to send.

The algorithms in this paper make use of ***collision detection***; players listening on the channel can distinguish between silence and noise. This means that, to any player listening in a slot, the channel provides trinary feedback: the slot is either silent, contains a successful broadcast, or is noisy. In the case of a successful transmission, any player listening on the channel can receive the message content. This is consistent with some prior results on contention resolution [18, 84, 85]

To enable high-probability results, we assume through the bulk of the paper that problem instances are ***γ-slack feasible***, by which we mean that it would be feasible to broadcast all messages by their deadlines while only using a constant γ fraction of the available channel bandwidth. Said differently, even if we multiply the length of each message by a constant $1/\gamma$, it should be feasible to broadcast each message by its deadline.

1.2 Results

We show how to solve contention resolution on a multiple-access channel when messages have arrival times and deadlines. Our protocol guarantees that for constant slack-feasible instances, each job j successfully broadcasts its message with high probability in its window size, that is, with probability at least $1 - 1/\text{poly}(w_j)$.[2] We find it surprising that these guarantees are possible for a constant slack γ. To simplify the presentation, we do not attempt to optimize the constants herein; it is an intriguing open problem to optimize γ further.

Outline: We begin by analyzing the algorithm UNIFORM, which we think is the most natural algorithm for contention resolution with deadlines—each job broadcasts once (or $\Theta(1)$ times) in a randomly chosen slot in its window. We show that UNIFORM is far from delivering the promised performance guarantees to jobs with deadlines. The good news about UNIFORM is that for sufficiently slack-feasible instances, with high probability in n, $\Theta(n)$ of the n messages get broadcast successfully. The bad news is that UNIFORM is not a fair strategy because it permits starvation: some jobs can have broadcast probabilities that are polynomially *small* in n. We explore the implications of UNIFORM in Section 2.

In contrast, we want each job to have a *large* success probability—the *failure* probability of a job should be polynomially small in its window size. To derive a better algorithm, we begin by focusing on the special case where the windows are ***power-of-2-aligned***, i.e., all the jobs' windows have a size that is a power of 2 and all the windows of size 2^ℓ arrive at a time that is a multiple of 2^ℓ.

Our basic strategy, called ALIGNED, for the power-of-2-aligned windows is to isolate jobs that share the same exact window, allowing them to coordinate collectively. Jobs in a window can quickly estimate the number of other contending jobs, and choose broadcast probabilities accordingly. However, windows of different sizes may interfere with each other. Thus, we mandate a simple rule: always defer to smaller windows. The contending jobs in some window of size w suspend their activity as long as there are any jobs in smaller windows concurrently trying to complete. In Section 3, we discuss how jobs measure contention, choose when to suspend their activity, and run their transmission protocol when they are active. Then we show that this procedure guarantees that each job completes its broadcast in its specified window with high probability (in the window size).

We show how to extend this approach to the general case where windows are *not* power-of-2-aligned and there is no global clock on which to synchronize. In this case, our PUNCTUAL protocol chooses a coordinator for each window that is responsible for synchronizing and allowing the jobs to simulate ALIGNED. At first glance, this approach seems circular, in that choosing a coordinator is no easier than successfully broadcasting a message. Selecting the coordinator leads to several algorithmic challenges. One of the ideas of PUNCTUAL is to separate two cases: either there are a large number of jobs and one should quickly select a coordinator and simulate ALIGNED, or there are not too many jobs and there is no need to run ALIGNED at all. One insight is that when contention is high, there are a lot of jobs, and hence one can quickly select a coordinator using low transmission probabilities without being likely to collide with other messages. We describe the PUNCTUAL protocol in Section 4.

[2]Notice that the probability of success necessarily depends on the window size, not the total number of jobs n, since most of those n jobs may be outside the relevant window and have no effect on the job in question.

2 GIVING INTUITION: CONTENTION AND WHY THE NATURAL ALGORITHM FAILS

In this section, we define the ***contention*** in a slot to be the sum of the broadcast probabilities of all of the jobs in the slot. We explain that in order for a slot to have a constant probability of successful transmission, the contention in the slot must be constant. We analyze the natural algorithm, UNIFORM, where each job broadcasts in one (or $\Theta(1)$) slots of its window and prove that UNIFORM successfully delivers $\Theta(n)$ messages with high probability. We demonstrate that UNIFORM fails as a credible protocol for two reasons. First, many jobs have super-constant contention for every slot in their window, and thus have a low probability of successfully broadcasting. Second, we want each job to succeed with high probability, and $\Theta(1)$ broadcast attempts is too little to achieve these guarantees.

Despite its flaws, UNIFORM succeeds in one important way: in developing intuition. In particular, UNIFORM serves as an example of what can go badly wrong, and it highlights the difficulties that our better algorithms must contend with.

2.1 Contention

Let $p_j(t)$ denote the probability that job j attempts to broadcast in slot t. We define the ***contention*** $C(t)$ in slot t to be $C(t) = \sum_j p_j(t)$, where the sum is taken over all jobs in the system at time t.

We employ the following well-known inequalities [84].

LEMMA 1 ([84]). *For any $0 \le x < 1$, $e^{-x/(1-x)} \le 1 - x \le e^{-x}$.*

For any fixed slot t, let $p_{\text{suc}}(t)$ denote the probability that some message is successful in slot t. We state the following results relating $C(t)$ and $p_{\text{suc}}(t)$; these have appeared in similar forms in [15, 28, 84]. In our algorithms for contention resolution with deadlines, no job ever sends in a slot with probability greater than $1/2$, which simplifies this relationship.

LEMMA 2. *If $p_i(t) \le 1/2$ for all i, then for any fixed slot t, the following holds:*

$$\frac{C(t)}{e^{2C(t)}} \le p_{suc}(t) \le \frac{2C(t)}{e^{C(t)}}.$$

COROLLARY 3. *If $p_i(t) \le 1/2$ for all i, then the following holds.*

- *If $C(t) = \Theta(1)$, then $p_{suc}(t) = \Theta(1)$.*
- *If $C(t) < 1$, then $p_{suc}(t) = \Theta(C(t))$.*
- *If $C(t) = \Omega(1)$, then $p_{suc}(t) = O(2^{-\Theta(C(t))})$.*

2.2 Analysis of Uniform: What Happens When You Broadcast Uniformly

This section analyzes the natural algorithm UNIFORM. We start with a desirable guarantee made by UNIFORM:

LEMMA 4. *When UNIFORM is run on a constant γ-slack feasible instance for $\gamma < 1/6$, a constant fraction of the messages broadcast successfully with high probability.*

PROOF. For starters, consider the case where job windows are power-of-2 aligned. That is, each window size w is a power of 2 and arrives at a time that is a multiple of w. Since each job makes a broadcast attempt independently, we can choose the order in which jobs flip coins to decide upon their broadcast slots. Given this, we consider a "revealing" process where we start considering the smallest job window, and then proceed with job windows of increasing size (breaking ties arbitrarily). Therefore, a job's broadcast slot can be ***temporarily clear***—that is, no job revealed so far has a collision—with a collision coming only later in the process. We refer to the broadcast as ***temporarily successful***.

Each time that a new job j randomly selects a slot to send in, the probability that j's broadcast collides with a broadcast attempt by a previously revealed job k that has already been assigned that slot is at most γ for a γ-feasible instance. In addition, if job j's broadcast attempt does result in a collision, it can ruin at most two transmissions. Job j's broadcast can collide with a broadcast from some previously revealed job k that was temporarily successful. Alternately, j could have chosen to broadcast in a slot that already contained a collision; in this case, j only ruins its own broadcast.

Define the indicator random variables $X_1, \ldots, X_n$, where

$$X_j = \begin{cases} 0 \text{ if } j \text{ chooses a slot that is temporarily clear,} \\ 1 \text{ if } j \text{ chooses a slot that already has a broadcast attempt.} \end{cases}.$$

The number of successful broadcasts is at least $n - 2\sum_{j=1}^{n} X_j$. By a standard Chernoff bound, and since $\gamma < 1/2$, the number of successful broadcasts is $\Theta(n)$, with probability at least $1 - \exp(-\Theta(n))$.

We now turn to the case when job windows are arbitrary rather than power-of-2 aligned. As before, we assign slots from smallest to largest. Now it is no longer the case that if the instance is γ-slack feasible, then this probability of a collision is at most γ. A weaker claim still holds, however—that the probability of a collision is at most 3γ. To see this, consider placing the next window W in the revealing process. Window W can overlap with previously-placed windows that: (1) are completely contained within W, (2) overlap either the starting or ending point of W. Given that we consider windows montonically increasing in size, the length of the interval spanned by windows in (1) and (2) is at most $3W$. Since the instance is γ-slack feasible, at most 3γ W slots contain a broadcast from the previously revealed jobs. Therefore, the probability of a collision is at most 3γ. Given this, and the assumption that $\gamma < 1/6$, the lemma holds. □

In contrast to Lemma 4 above, we now show that UNIFORM possesses the following unpleasant property: it is not the case that each message has (at least) a constant probability of transmitting—in fact, some messages are overwhelmingly likely to fail.

The problem with UNIFORM is that messages do not get a fair shot at accessing the channel. Indeed, some messages have almost no chance at successfully transmitting. Ironically, the high-priority messages—those that have small window sizes and therefore must be executed soon—are most at risk of starving.

LEMMA 5. UNIFORM *is not a fair strategy. That is, for every γ, there exist γ-slack-feasible instances such that certain jobs have a probability of $O(1/n^{\Theta(1)})$ of broadcasting successfully.*

PROOF. Consider the γ-slack-feasible instance where all n jobs arrive at the beginning of slot 1, and each job j has window size $w_j = j/\gamma$ for $j \ge 1$. Given that jobs execute UNIFORM, $C(1)$ is order the harmonic number, $H(n) \approx \ln(n)$. Indeed, for $1 \le t \le n^{1-\delta}$, for any constant $0 < \delta < 1$, $C(t) \ge \delta \ln(n) - O(1)$. By Lemma 2, the probability that any of the first $\Omega(n^{1-\delta})$ jobs have a successful broadcast is $O(\ln n / n^{1-\delta})$. □

In summary, offering a strong guarantee of success for all packets is certainly out of reach for UNIFORM. Transmitting once per window, if you do it randomly, is bad, since there is too much contention. If we want high-probability bounds, we must transmit more than a constant number of times per window which generates even more contention, and many more jobs would starve as a result.

The bottom line is that UNIFORM is a great algorithm for developing intuition, but a dead-end as a way to achieve our desired performance guarantees. In subsequent sections, we provide nontrivial algorithms that *do* succeed in guaranteeing that each job transmits its message with high probability in its window size.

3 SPECIAL CASE: HANDLING ALIGNED WINDOWS

In this section, we give an algorithm, ALIGNED, designed for the special case where all windows are power-of-2-aligned; recall that this means that each window size w is a power of 2 and arrives at a time which is a multiple of w. This temporary assumption enables us to zoom in on a crucial subroutine of our general algorithm, PUNCTUAL, which we finish presenting in Section 4.

Our algorithm uses a form of *pecking-order scheduling* [11, 12], which prioritizes jobs based on window size. That is, jobs in larger windows should generally yield to jobs in smaller windows. If there were a centralized scheduler that could perform pecking-order scheduling globally, then at least for instances where all job deadlines are different, it is not hard to see that the centralized scheduler could give one slot to each job. In fact, such a schedule would simply be earliest-deadline first, which is optimal for scheduling jobs with deadlines. If multiple jobs have the same window, but the centralized scheduler cannot schedule at the granularity of individual jobs, the problem becomes a bit more complex—we need a contention resolution protocol among jobs having the same window. Nevertheless, developing such an algorithm is not too hard.

A challenge is that we do not have a centralized scheduler that can perform pecking-order scheduling globally. Instead, the jobs need to coordinate in a distributed manner. When a job arrives, it has a dilemma: the job does not know a priori whether most other jobs currently in the system are larger or smaller than it. If most of the jobs it is contending with have smaller windows, then it should defer to them; otherwise, then it should transmit more aggressively.

Here we make use of the synchronization implicit to aligned, laminar windows. (In Section 4, we remove this assumption and tackle the new challenges that arise.) Specifically, the window boundaries themselves trigger transitions between jobs with different windows, allowing jobs to coordinate their pecking order. ***Job class ℓ*** refers to a set of jobs with the same window W whose size is $w = 2^\ell$. We call each timestep that marks the beginning of a window of job class ℓ a ***critical time*** for that job class.

Algorithm for Power-of-2-Aligned Windows

The main idea of the algorithm is as follows. At any time, only the jobs in one job class, say job class ℓ, are actively participating and attempting to transmit on the channel. We call this job class the ***active job class***, and we say that all jobs in this class are ***active jobs***. All jobs in other job classes simply listen and wait their turn; they do not actively participate. When a critical time is reached for any smaller job classes, the smallest such job class becomes the active one, and the algorithm for job class ℓ is suspended. Eventually, the smaller job classes complete their jobs, thus yielding control back to job class ℓ, and those larger jobs continue their algorithm where they left off.

We first outline the algorithm for a single job class ℓ. In each timestep that this job class is active, i.e., in an ***active step***, all jobs in the class perform the next transmit/listen step of the algorithm:

(1) *Estimation:* Use $T_\ell = \Theta(\text{poly}(\ell))$ active steps to estimate the number of jobs in job class ℓ currently in the system. Let n_ℓ denote this estimate.
(2) *Broadcast:* Use $\Theta(n_\ell + \text{poly}(\ell))$ active steps to broadcast the messages for jobs in job class ℓ.
(3) *Truncation:* If there are any remaining jobs in this class, they give up and implicitly yield control back to the larger job classes. We shall show that, with high probability (in 2^ℓ, the window size), all jobs in the class complete.

Observe that the total number of active steps used by this algorithm is $T_\ell + \Theta(n_\ell + \text{poly}(\ell))$, which depends on the estimate of the number of jobs in the class.

When a job in class ℓ enters the system at its release time, the job is considered ***live***. The job remains live until either (1) the above algorithm has taken enough steps to run to completion, or (2) the end of the job's window is reached, whichever happens first. If the end of the job's window is reached first, we say that the algorithm is ***truncated***. Only live jobs run the algorithm.

Jobs run this algorithm in active steps, i.e., when their job class is active. But all jobs also need to agree on which job class is active at any particular time. To do so, all jobs passively simulate any lower-level protocols by waiting the appropriate number of steps and listening on the channel. That is, suppose a job class ℓ becomes active at time t (t is critical for ℓ). Then all live jobs (the only live jobs are in the larger windows) also begin to simulate the algorithm for job class ℓ. At each step, all jobs simulate the next step for the smallest class whose algorithm has not yet completed. By listening on the channel, the live jobs all see the result n_ℓ of the estimation protocol for class ℓ and therefore know how long the simulation should last.

Figure 1 gives an example schedule of active steps, which consists of windows of three sizes (small, medium, and large). As shown here, the first small, medium, and large window each need 4, 7, and 7 active steps to complete their algorithms. These active steps are scheduled as early as possible, with priority given to the smallest window. That is, the first 4 steps are active for the first small window. The next 5 steps are active for the medium window; this is not enough steps to complete the algorithm, but when the second small window begins, it becomes active. After 2 active slots for the second small window, the medium window can resume and finish.

Given that smaller windows/classes always have priority, it is possible that some (large) class may not be able to schedule all of its active slots before the end of its window and be truncated. To argue that the algorithm works well, it is important that truncations be unlikely. Informally, this is true because with sufficient slackness (small enough γ), $T_\ell + \Theta(n_\ell + \text{poly}(\ell)) \ll 2^\ell$. Thus, some time remains in the window during which larger classes can complete their protocols.

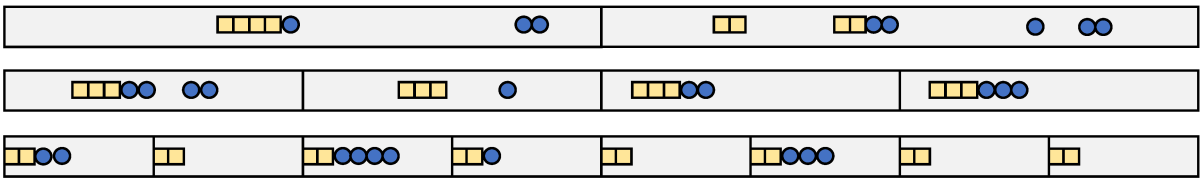

Figure 1: An example of pecking-order scheduling for aligned windows. Time is displayed horizontally, and each row corresponds to windows of a single size. The yellow squares indicate active timesteps during which the estimation procedure is performed, and the blue circles indicate active timesteps during which data messages may be transmitted. The first large window is thus active for a total of 7 timesteps. Notice that these active steps need not be contiguous, as lower windows have higher priority. The number of blue circles is proportional to the estimate on the number of jobs in the window, so in this example the estimate in the first window in the bottom row is roughly half as large as the estimate in the third window in that row.

Details for a Single Job Class

There are two details yet unspecified for a job class: (1) how to estimate the number of jobs in the window, and (2) what algorithm to use to broadcast data messages. Throughout, there are constant parameters to the algorithm that affect failure probability. To reduce the proliferation of variables, we use λ in multiple places to indicate a parameter that affects the failure probability. Each occurrence of λ could in principle be tuned to a different value, but we use the single parameter for simplicity.

Size-estimation protocol. We first describe a simple estimation protocol for job class ℓ. This protocol uses $T_\ell = \lambda\ell^2$ timesteps.

- The protocol is divided into ℓ phases, $1, 2, 3, \ldots, \ell$. Each phase consists of $\lambda\ell$ steps.
- During each step of the ith phase, each job in the job class transmits a control message with probability $1/2^i$.
- Throughout the process, jobs listen to the channel and count the number of successful transmissions.
- After completing all phases, let j be the phase with the most successful transmissions. Then the estimate is $n_\ell = \tau \cdot 2^j$, for some constant $\tau > 1$ to be specified later.

Note the factor of τ in the estimate. This is to ensure that the estimate is likely to be an overestimate rather than an underestimate, which simplifies the analysis of the broadcast algorithm.

For concreteness, if the algorithm is truncated during or before the estimation component, we resolve the estimate to 0.

Broadcast. We now describe the broadcast protocol for jobs in class ℓ, assuming we have already found an estimate n_ℓ. (Observe that n_ℓ is always a power of 2.) Since only jobs in class ℓ are taking steps, and we have an estimate on the number of jobs, we can treat this as a *batch* problem. This yields a relatively simple backon protocol:

- The algorithm is divided into $\log_2 n_\ell + \ell$ ***phases*** numbered $0, 1, \ldots, \log_2 n_\ell + \ell - 1$. For $i < \log_2 n_\ell$, the length of the ith phase is $\lambda n_\ell / 2^i$ timesteps. Each of the final ℓ phases has length $\lambda\ell$ timesteps.
- Consider a phase with length λX. This phase is divided further into λ ***subphases***, each of length X. During each subphase, each job in the job class chooses a random timeslot in which to transmit its data message. If the transmission is successful, the job terminates its algorithm.

The main idea behind the analysis is the following. If the number of live jobs remaining is smaller than the phase length X, then a constant fraction of the jobs are likely to complete in that phase. In particular, the failure probability is exponentially small in X. But we want the failure probability to be polynomially small in the job's window size (exponentially small in ℓ), which means that we cannot rely on successes whenever $X \ll \ell$. The extra ℓ phases of length $\lambda\ell$ at the end enable us to prove a high-probability result throughout.

Now that the algorithm is defined, we can state the number of active steps required. The constants here are not important except insomuch as they exist. Specifically, the number of active steps needed can be determined directly from the class ℓ and the estimate.

Lemma 6. *Consider a job class ℓ, and let n_ℓ be the result of the estimation procedure. Then the total number of active steps needed to complete the algorithm for this job class is $2\lambda(\ell^2 + n_\ell - 1)$.*

Proof. The estimation algorithm trivially uses $\lambda\ell^2$ steps. The broadcast algorithm has phases of length $\lambda n_\ell, \lambda n_\ell/2, \ldots, \lambda 2$ followed by ℓ phases of length $\lambda\ell$ each. The total length of the broadcast algorithm is thus $\lambda(\sum_{i=1}^{\lg n_\ell} 2^i + \ell^2) = \lambda(2n_\ell - 2 + \ell^2)$. □

Jamming. It turns out the algorithm we have described can also handle a sort of stochastic "jamming" wherein an adversary can selectively prevent successes in some rounds. We believe that this is an interesting property of the approach presented, and so for the purpose of this section only, we will consider performance with respect to this stochastic noise.

Specifically, assume there is an adversary that can look at slots and decide to create noise in that slot, e.g., if a message is broadcast. (Here the adversary can even look at the contents of the message itself.) If the adversary decides to jam, the jamming succeeds with some constant probability p_{jam}. We allow the adversary to choose whether to jam as it is conceivable that selectively jamming actually makes the algorithm perform worse. For example, the adversary could conceivably skew the estimate n_ℓ by jamming only some of the phases during the estimation protocol.

Analysis of Aligned Algorithm

The analysis involves several components. First, we argue that all jobs agree on the schedule of active slots, which is important to ensure that different job classes do not interfere with each other. Given that, the remainder of the analysis focuses on the algorithm for individual job classes. The second component of the analysis is proving that the size estimation is accurate to within a constant factor. Third, we argue that algorithms are unlikely to be truncated. Finally, we argue that the during the broadcast phase of the algorithm, all jobs are likely to successfully transmit their data messages.

Lemma 7. *For every time slot t, every live job j agrees on which job class is active at time t.*

Proof. The proof is by induction over time. Consider every job class ℓ. The claim is more precisely that at time t, all live jobs in classes $\ell' \geq \ell$ (1) know exactly how many active steps class ℓ has taken, and (2) have seen the same results of ℓ's possibly ongoing size estimation protocol. It follows that if the estimation has completed for ℓ, then all larger job classes also know exactly how many more active steps ℓ needs before finishing.

Now consider what happens at time $t+1$. If $t+1$ marks the end of a window for some job class ℓ, then all job classes $\ell' \geq \ell$ observe the window boundary and agree that ℓ's execution is now truncated. If $t+1$ is critical (the start of a window) for some job class ℓ, then all job classes $\ell' \geq \ell$ observe the window boundary and agree that ℓ is on the first step of its protocol. It is now just a matter of taking or simulating the next active step. Let ℓ_0 be the smallest job class that has not completed its algorithm. Then all live jobs are in classes $\geq \ell_0$, and they all agree that ℓ_0 should take one active step. □

We next turn our attention to the size estimation algorithm. We focus on proving the following lemma, which says that the estimate is likely to be within a constant factor of the true number of jobs in the class.

Lemma 8. *Let $\hat{n}_w$ denote the number of jobs with a particular window of size $w = 2^\ell$, assume that $p_{\text{jam}} \leq 1/2$. There exists a sufficiently large constant τ (a parameter in the algorithm) such that if the size-estimation protocol for the window completes, then with probability at least $1 - 1/w^{\theta(\lambda)}$ the estimate n_w satisfies $2\hat{n}_w \leq n_w \leq \tau^2 \hat{n}_w$.*

Before proving the lemma, it helps to recall the size estimation algorithm. The estimation procedure is divided into ℓ phases. In the ith phase, each job transmits in each slot independently with probability $1/2^i$. To prove the lemma, we first prove two bounds on individual phases. The first says that if $2^i \approx \hat{n}_w$, then there are likely to be many successful transmissions during the phase. The second says that if $2^i \not\approx \hat{n}_w$, then there are not likely to be many successful transmissions.

Lemma 9. *Let $\hat{n}_w \geq 1$ denote the number of jobs with a particular window of size $w = 2^\ell$, and assume that $p_{\text{jam}} \leq 1/2$. Consider the i-th phase of the size estimation algorithm, assume that the phase is not truncated, and suppose that $2^{i-1} \leq \hat{n}_w \leq 2^i$. Then with probability at least $1 - 1/w^{\Theta(\lambda)}$, the number of successful transmissions during the phase is at least $\lambda\ell/16$.*

Proof. First consider what happens without jamming and let us calculate the probability of a successful transmission in each slot in the phase. Recall that each job transmits with probability $1/2^i$. The probability that exactly one job transmits is $\hat{n}_w(1/2^i)(1-1/2^i)^{\hat{n}_w - 1} \geq 2^{i-1}(1/2^i)(1-1/2^i)^{2^i-1} \geq 1/(2e)$.

With jamming, the probability of a successful transmission in each slot may be reduced by half, to at least $1/(4e)$.

We can now take a Chernoff bound over all slots to get the number of successful transmissions. The expected number is at least $\lambda\ell/4e$, so with probability at least $1 - 1/w^{\Theta(\lambda)}$, the number of successes is at least $\lambda\ell/16$. □

Lemma 10. *Let $\hat{n}_w$ denote the number of jobs with a particular window of size $w = 2^\ell$. Consider the i-th phase of the size estimation algorithm and suppose that either $\hat{n}_w \leq 2^{i-5}$ or $\hat{n_w} > 2^{i+5}$. Then with probability at least $1 - 1/w^{\Theta(\lambda)}$, the number of successful transmissions during the phase is strictly less than $\lambda\ell/16$.*

Proof. As in Lemma 9, we look at the probability of a successful transmission in each slot. If $\hat{n_w} \leq 2^{i-4}$, then the probability that at least one job chooses to transmit (which upper bounds success) is at $2^{i-5}/2^i \leq 1/32$. If $\hat{n_w} \geq 2^{i+4}$, then the probability of a successful transmission is $\hat{n}_w(1/2^i)(1-1/2^i)^{\hat{n}_w - 1} \leq \hat{n}_w(1-1/2^i)^{\hat{n}_w} \leq \hat{n}_w(1/e)^{\hat{n}_w/2^i} = r/e^r$ for $r = \hat{n}_w/2^i$. Assuming $r > 2^5$, this value is at most $1/32$. The expected number of successful transmissions is thus at most $\lambda\ell/32$. To conclude, we can apply a Chernoff bound across all slots to get that with probability at most $1 - 1/w^{\Theta(\lambda)}$, the number of successes is less than $\lambda\ell/16$. □

We are now ready to complete the proof of Lemma 8.

Proof of Lemma 8. Lemma 9 states that for phase i with $2^{i-1} \leq \hat{n}_w \leq 2^i$, the number of successes is likely to be at least $\lambda\ell/16$. Lemma 10 states that for phases with $2^i \geq 32\hat{n}_w$ or $2^i \leq \hat{n}_w/32$, the number of success is likely to be smaller than $\lambda\ell/16$. If neither bound fails, the maximum must fall in a phase j that satisfies $\hat{n}_w/32 \leq 2^j \leq 32\hat{n}_w$. Choose $\tau = 64$, giving estimate $n_w = \tau 2^j$, to complete the proof. □

Thus far, we have shown that the size estimation works well. For the remainder of the paper, we assume that τ is fixed to satisfy Lemma 8 and omit the assumption from subsequent lemma statements.

We next turn to scheduling the active steps. Our goal is to prove that the algorithms are unlikely to be truncated. That is, we wish to show that for particular job class ℓ, all nested windows do not have too many active slots in aggregate, and thus enough unclaimed slots remain to complete ℓ's algorithm.

Lemma 11. *Consider a particular window W with size $w = 2^\ell$. Let $\hat{N}_W$ denote the total number of jobs whose windows are nested inside W, including those jobs with window exactly W. Let w_0 denote the smallest window size. For sufficiently large λ, with probability at least $1 - 1/w^{\Theta(\lambda)}$, the sum of all estimates is at most $2\tau^2\hat{N}_W + 2w/w_0$.*

Proof. Note that if the size estimation is truncated, then the estimate is 0. Since we want an upper bound here, the worst case is to assume that all estimates complete.

We divide the windows into two groups: those that are larger than $\sqrt{w}$, and those that are smaller than $\sqrt{w}$. Let $\hat{N}_B$ and $\hat{N}_S$ denote the number of jobs in each group, respectively. We have $\hat{N}_B + \hat{N}_S = \hat{N}_W$. Let N_B and N_S denote the sums of the estimates produced for windows in each group.

We can bound N_B by simply taking the union bound across all windows and applying Lemma 8. The important point is that for any $\bar{w} \geq \sqrt{w}$, $w^{\Theta(\lambda)} = \bar{w}^{\Theta(\lambda)}$, and hence the high probability results are sufficient. We thus have that $N_B \leq \tau^2 \hat{N}_B$.

Next consider the small windows. Some small windows have successful estimates that fall below the threshold in Lemma 8. Their total sums to at most $\tau^2 \hat{N}_S$. The remainder of the proof focuses on bounding the contribution of failures. In particular, showing that with probability at least $1 - 1/w^{\Theta(\lambda)}$, the sum of all estimate failures is at most $2w/w_0$.

Consider all the windows of one particular size $\bar{w} < \sqrt{w}$, and let $k = w/\bar{w}$. It suffices to show that the total of failed estimates for these windows is at most $w/\bar{w}$. Then summing across all sizes then yields a total of at most $\sum_{\ell=\log(w_0)}^{\infty} w/(2^\ell) \leq 2w/w_0$. Our goal is thus simply to show that it is unlikely for there to be more than $k/\bar{w}^2$ failures for this size. By construction, any estimate is at most $\bar{w}$, so the total $k/\bar{w}^2$ failed estimates is at most $k/\bar{w}$ as desired.

It remains to bound the number of failures for windows of size $\bar{w}$. Observe that for sufficiently large λ, the probability of a failure in Lemma 8 is at most $1/(2\bar{w}^2)$. Thus, the expected number of failures is at most $k/(2\bar{w}^2)$. Because $k \geq \sqrt{w}$ and each window's failure is independent, we can apply a Chernoff bound to conclude that with probability at most $1 - 1/w^{\Theta(\lambda)}$, the total number of failures is at most $k/\bar{w}^2$ as desired, completing the last step of the proof. □

We next argue that active steps can be scheduled.

Lemma 12. *For all λ, there exists a sufficiently small γ such that the following holds. Suppose that the instance is γ slack feasible. Consider any window W of size w. Then with probability at least $1 - 1/w^{\Theta(\lambda)}$, the algorithm for this window runs to completion, i.e., it is not truncated.*

Proof. Note that being γ slack feasible implies that every job window has size at least $w_0 \geq 1/\gamma$. Let $\hat{N}_W$ denote the total number of jobs with window W or nested windows, and let N_W be the sum of all estimates for these windows.

Our goal is to bound the total number of active slots desired by W and all nested windows. As long as the total is smaller than w, then all of W's active slots can be scheduled. (For this proof, we do not care if a smaller window's algorithm gets truncated — all that matters is how much it can get in the way of W.)

By Lemma 6, the total number of active steps for W and all nested windows is at most $2\lambda \sum_{\ell=\log(1/\gamma)}^{\log w} \ell^2 + 2\lambda N_W$. We simply need to upper bound each term by say $w/2$. By Lemma 11, the second term is appropriately bounded, with probability at least $1 - 1/w^{\Theta(\lambda)}$, as long as (1) $\gamma < 1/(16\lambda\tau^2)$, giving $2\tau^2 \hat{N}_W \leq 2\tau^2(\gamma w) \leq w/(8\lambda)$, and (2) $1/\gamma \geq 16\lambda$, giving $2w/w_0 \leq w/(8\lambda)$. The summation term is deterministic, and it solves to at most cw for some constant that depends on the smallest value of ℓ; there exists a small enough γ to drive c down to $1/(4\lambda)$. □

Given that the active steps can be scheduled, the last component of the proof is to show that the data messages are successfully transmitted.

Lemma 13. *Consider a window W of size $w = 2^\ell$, let $\hat{n}_w$ denote the number of jobs in the window, and let n_w denote the result of the estimation procedure for the job class. Suppose that $p_{\text{jam}} \leq 1/2$, $n_w \geq 2\hat{n}_w$, and that the broadcast algorithm for W is not truncated. Then with probability at least $1 - 1/w^{\Theta(\lambda)}$, all jobs with window W succeed in broadcasting their data messages.*

Proof. Recall that the algorithm consists of phases of length $\lambda n_w, \lambda n_W/2, \ldots, \lambda 2$ followed by ℓ phases of length $\lambda\ell$. For the decreasing phases, we shall only leverage those with long-enough lengths $\lambda n_w, \lambda n_w/2, \ldots, \lambda\ell$, ignoring the smaller phases. We shall argue that (1) during these decreasing phases, the number of jobs is likely to be reduced to at most $\ell/2$, and (2) if the final phases begin with at most $\ell/2$ jobs, then they are likely to all finish.

For the decreasing phases (down to length $\lambda\ell$), the proof is by induction over phases. Let $X_i = n_w/2^i$. The inductive claim is that at the start of phase i, the number of live jobs is at most $X_i/2$ with probability at least $1 - 1/w^{\Theta(\lambda)}$. The claim holds initially as $\hat{n}_w \leq n_w/2 = X_0/2$. Now we prove the inductive step. Each subphase has length X_i, and each job chooses one random slot in which to transmit. Because the number of jobs is bounded, the probability of a collision for the job is at most $1/2$. Assuming no collision, the probability of being jammed is at most $1/2$. So the job successfully transmits it data message with probability at least $1/4$. Repeating λ subphases means that the probability of the job failing to transmit is at most $(3/4)^\lambda$. Thus, by a Chernoff bound, the probability that more than $X_i/4 = X_{i+1}/2$ jobs fail is at most $1 - 1/e^{\Theta(\lambda X_i)} \geq 1 - 1/w^{\Theta(\lambda)}$, where the inequality follows because $X_i = \Omega(\log w)$.

For the equal phases, the proof is easier. In each subphase, each job fails with probability at most $3/4$ as before. So the probability that the job fails in all $\lambda\ell$ subphases is at most $(3/4)^{\lambda\ell} = 1/w^{\Theta(\lambda)}$. Taking a union bound across all jobs completes the proof. □

Finally, we combine all the components to attain the main theorem.

Theorem 14. *For all λ, there exists a sufficiently small γ such that the following holds. Consider a γ slack-feasible instance, let j be any job and let w be its window size. Then with probability at least $1 - 1/w^{\Theta(\lambda)}$, job j successfully transmits its data message.*

Proof. By Lemma 12, the job class is likely to complete its algorithm with probability at least $1 - 1/w^{\Theta(\lambda)}$. By Lemma 8, $n_w \geq 2\hat{n}_w$ with probability at least $1 - 1/w^{\Theta(\lambda)}$. Thus, the assumptions of Lemma 13 are likely to hold, and we conclude that with probability at least $1 - 1/w^{\Theta(\lambda)}$ that the data message is successfully transmitted. □

4 PUNCTUAL BACKOFF: HANDLING GENERAL WINDOWS

In this section, we fulfill the promise that we made in the introduction. Now supplied with the necessary tools, we set out to solve our original problem: contention resolution with deadlines. We give an algorithm, PUNCTUAL, which, true to its name, guarantees the

following. For any instances with sufficient slack, each job successfully broadcasts within its window, with high probability in the job's window size.

Trimming down windows. We begin by giving the impetus for considering the aligned-window special case from Section 3. For this, we need terminology. If W is an arbitrary window, we say that $\text{trimmed}(W)$ is a largest aligned window that is contained in W; if there is more than one largest window, choose arbitrarily. Notice that $|\text{trimmed}(W)| \geq |W|/4$. If J is a set of jobs, then $\text{trimmed}(J)$ is the set of jobs in which the window W associated with each job is replaced with $\text{trimmed}(W)$.

We employ an analog to Lemma 6 in [12]:

Lemma 15 ([11, 12]). *Consider any 4γ-slack feasible set of jobs J, where $1/\gamma$ is an integer. Then, $\text{trimmed}(J)$ is γ-slack feasible.*

With enough slack feasibility, a global scheduler can convert any scheduling instance to one that is power-of-2 aligned, and then use pecking-order scheduling on that instance as described in Section 3. In addition, if all jobs had access to a global clock—that is, all jobs agreed on the index of the current slot—then each job could trim its own window without any help. Then, the algorithm from Section 3 could be used. However, a global clock is not available here. (For work addressing the power of a global clock, see [38, 69].)

The contention dilemma again. We still face the same dilemma as in Section 3. That is, when a job enters the system, it must learn whether its window is smaller or larger than those of other jobs in the system. A job with a smaller window should aggressively attempt to seize the channel, while a job with a larger window should remain silent so that other jobs can succeed.

The aligned solution from Section 3 only worked because there existed critical times. Each critical time for a job class triggered an estimation protocol, giving all jobs in the system the same estimate of the number of jobs in that class. These estimation protocols resolved the dilemma.

But now we no longer have these critical times. Now that job arrival times and window sizes are arbitrary, it may feel that we are starting all over from scratch.

Choosing a leader. We want to resolve the situation by somehow choosing a ***leader*** to broadcast a global time to everyone in the system. In particular, if job j is elected leader, then for the rest of its window, it broadcasts (its own version of) a global time in every other time slot. Then j broadcasts its own message at the end of its window and terminates. At this point, the system is leaderless and the jobs must collectively identify a new leader.

Whenever the system is running with a leader j, there is a shared global time. Therefore, for any job that has an earlier deadline than the leader, there are critical times which can be used for running estimation protocols.

How can a job help elect a leader? How can it do so without blocking jobs from other job classes?

Why the contention dilemma is an obstacle to choosing a leader. Now the contention problem once again shows its ugly head.

When job j arrives, it first listens to determine whether there is already a leader in the system that it can look to. If not, j needs to participate in a leader-election protocol for its job class.

We seem have a Catch-22. The job class needs to broadcast on the channel to run its protocol. But if the probability that a job j makes a broadcast attempt is $\Omega(1/w_j)$, then it contributes too much contention, and jobs from smaller size classes get starved; see Section 2.2. On the other hand, if the probability that job j makes a broadcast attempt is $o(1/w_j)$, then there is a super-constant probability that j never broadcasts during its entire window.

To summarize, a leader must be elected so that both small-window and large-window jobs get their fair shot at the channel. But in order to run a leader-election protocol, the jobs also need to get their fair shot at the channel to run the protocol.

A slingshot solution. But the Catch-22 has a catch. Occasionally there are job classes that do not need a leader in order to broadcast successfully, while not causing too much damage with their libertarian ways.

To exploit this catch, jobs must be able to recognize when a leader is not required. The idea is to essentially have *two size-estimation protocols*. The first one is implicit in the leader election itself, and the second one occurs after leader election. This idea is not nonsensical.

The first size-estimation protocol is rough and helps to determine when a job does not need a leader. By this coarse estimation, either a leader is identified, or the protocol estimates that there is a relatively small number of jobs in the job class and a leader is superfluous.

Job j's behavior is like a slingshot; first it pulls back, and then it decides whether or not to release.

Rounds and slots. To implement this slingshot mechanism, we isolate the activity of the jobs that are in different parts of the protocol. To this end, we group time slots into ***rounds***. Specifically, each round consists of four time slots: (i) a ***timekeeper*** slot, where leaders broadcast, (ii) an ***aligned*** slot, where the batch protocols run, and two slots for the slingshot protocol: (iii) a ***leader-election*** slot, where leaderless jobs broadcast when trying to elect a leader, and (iv) an ***anarchist*** slot, where jobs broadcast when they have given up trying to identify a leader.

How do jobs agree on which time slots are used for which purposes? In this case, we add six synchronization slots to each round: the first two slots of a round are ***synch*** slots used to broadcast ***start*** messages (and may consist of collisions); after these two synch slots, we alternate empty ***guard*** slots with the four useful slots described above.

This synchronization scheme ensures that the only time that a job observes two collisions or messages in a row is during the first two start slots of a round. Thus, when a job enters the system, it waits until it hears two consecutive slots with messages or collisions. If after 10 slots, this does not occur, then the job can itself broadcast start messages and begin a new round.

In total, this synchronization mechanism takes only $O(1)$ time, and uses up a constant fraction of the available window.

From this point on, we assume that jobs have synchronized their rounds, and we describe the protocol in terms of how they behave in the different types of slots.

How to use the slingshot. When a job j arrives, it first does two preliminaries: it establishes its round synchronization, and it rounds down its window size to the nearest power of 2. (This decreases the slack by at most a factor of 2.)

Next, it listens for a single slot in the timekeeper slot to see if there is a leader. If there is no leader, or if the leader has an earlier deadline than the deadline of job j, then it moves on to run the slingshot protocol. Otherwise, it happily moves to the aligned slot, trims its window based on the leader's time announcements, and runs the batch protocol from Section 3 using the aligned windows indicated by the leader's timestamps.

If a job j moves to the slingshot protocol, then it transmits with a (low) probability of $1/(w_j \text{polylog}(w_j))$, repeatedly, in the leader-election slot with the goal of claiming leadership; when it does not broadcast, it listens for other claimants.[3] This is the ***pullback stage*** of the slingshot algorithm. If job j becomes the leader, then it takes over the job of being the timekeeper/leader, broadcasting the round number (and its own deadline) in every timekeeper slot. Even if there is an existing leader, j necessarily has a later deadline (or it would not have started the leader-election process in the first place), and so the old leader steps aside for j.

If a leader emerges (whether j or someone else) with a deadline after that of j, then job j can move directly to the aligned slots, trims its window based on the leader's time announcements, and runs ALIGNED.

Otherwise, if no leader emerges after a $\text{polylog}(w_j)$ number of steps of the pullback stage, then j checks the current leader again, listening in the next timekeeper slot. If the current leader has a deadline after $1/2$ of j's deadline, then that is good enough: j rounds down its window by a factor of 2 (so now its deadline is before the current leaders), and again job j moves to the aligned slots, trims its window based on the leader's time announcements, and runs ALIGNED.

Finally, if no leader emerges after a $\text{polylog}(w_j)$ number of steps of the pullback stage and/or the current leader's deadline is too early, then j releases the slingshot, moving to the anarchist slot and substantially increasing its sending probability to $\Theta(\log(w_j)/w_j)$ for the remainder of its window. This is the ***release stage*** where the slingshot fires.

We will show that if there are $O(w_j/\log^3 w_j)$ jobs in the job class, then these will have a small effect on the contention in the anarchist slot, even when they are aggressive about trying to access the channel (in the release stage). Alternatively, if there are $\Omega(w_j/\log^3 w_j)$ jobs in the job class, then the protocol identifies a leader in $O(\text{polylog}(w_j))$ steps, and so stays away from the anarchist slot.

Proof of the General Algorithm

The bulk of this analysis focuses on anarchists — we already know that those jobs that follow a leader have a high probability of success.

LEMMA 16. *For any constant $\varepsilon > 0$, there exists a sufficiently small γ such that: if the instance is γ slack feasible, then the contention in every leader-election slot is at most ε.*

[3]This is such a low probability that if it kept competing to be the leader for the entire window, j would most likely not transmit even once.

PUNCTUAL for job j:

- When job j enters the system, it first rounds its deadline d_j down to the closest power of 2 and checks whether the slots are already synchronized into rounds. To do so, it waits until it hears two consecutive slots with messages or collisions. If after 10 slots, this does not occur, the j runs SYNCHRONIZE. From this point on, j always broadcasts start messages in the first two slots of every round.
- Job j listens on the timekeeper slot. If j hears a leader with a deadline $\geq d_j$, then it runs FOLLOW-THE-LEADER; if not it runs SLINGSHOT.

SYNCHRONIZE for job j:

- Job j broadcasts two start messages. Regardless of whether these two start messages collide, now the system is synchronized into rounds.

SLINGSHOT for job j:

- Repeat $\lambda \log^7(w_j)$ slots or until a leader is elected.
  - In each leader-election slot, with probability $1/(w_j \log^3(w_j))$, transmit "I am the leader with deadline d_j."
  - If j successfully transmits, it becomes the leader.
  - If j hears a j' successfully transmit in the leader-election slot, and $d_{j'} \geq d_j$, then j' becomes the leader and j runs FOLLOW-THE-LEADER.
  - If j hears a j' successfully transmit and become a leader, but $d_{j'} < d_j$, then j still continues running SLINGSHOT.
- If after these $\lambda \log^7(w_j)$ steps, j still has no leader, then it listens again in the timekeeper slot and if there is a leader with deadline at least $d_j/2$, then job j rounds its deadline down to $d_j/2$ and runs FOLLOW-THE-LEADER. Otherwise, for the rest of its window or until it succeeds, job j transmits its data message in each anarchy slot with probability $\lambda \log(w_j)/w_j$.

FOLLOW-THE-LEADER for j:

- Job j learns of the current leader and the global time in the timekeeper slots.
- Job j trims its window according to the global time and executes ALIGNED in the aligned slots.

BECOME-LEADER for j:

- In every timekeeper slot of the window, j sends "I am the leader", the value of d_j, and the current time.
- In the last timekeeper slot of j's window, j broadcasts its data message along with the current time and a message "I am abdicating."
- If j is deposed, then, in the next timekeeper slot, the old leader broadcasts its data message along with the global time, and then the new leader takes control of the timekeeper slots.

Figure 2: Pseudocode for PUNCTUAL.

Proof. First consider the contention from each job j in the system whose window size $w_j \in [2^\ell, 2^{\ell+1})$. There are at most $\Theta(2^\ell)$ such jobs in the system at any one time. In the pullback phase of SLINGSHOT, each such job broadcasts with probability $\Theta(1/(\ell^3 2^\ell))$, and thus these jobs contribute $O(1/\ell^3)$ to the contention. Therefore, the total contribution to the contention of jobs in any slot of the pullback phase of SLINGSHOT is $O\left(\sum_{i=1}^{\infty} \ell^{-3}\right) = O(1)$. □

Consider any particular job j. We say that leader election is ***successful*** for job j if one of the following two cases applies: (1) when job j arrives, there is already a leader with deadline later than j's deadline, or (2) job j successfully transmits a message during the leader-election component of SLINGSHOT. Observe that if case (2) occurs, then the resulting leader has deadline at least as late as j's deadline.

Lemma 17. *Consider any set S of jobs having window size w. Suppose $|S| \geq w/\log^3 w$. If the instance is γ slack feasible for sufficiently small constant γ, then with probability at least $1 - 1/w^{\Theta(\lambda)}$ at least one job in S has a successful leader election.*

Proof. Consider any $j \in S$. If there is already a leader with a later deadline, then j's leader election is successful. Assume for the remainder that no such leader exists when j arrives.

We say that job j has a successful leader election at time t if (1) j transmits at time t, and (2) there is no other transmission at time t. By Lemma 16 with $\varepsilon = 1/2$, (2) holds with probability at least $1/2$. And trivially (1) holds with probability $1/w\log^3 w$ because that's the transmission probability. Let X_{tj} be an indicator that job j has a success at time t. We thus have that $\Pr[X_{tj} = 1] \geq 1/(2w\log^3 w)$. Let $X = \sum_{j\in S}\sum_t X_{tj}$ denote the total number of successes from S. Observe that $E[X] \geq \lambda \log w/2$ as each job is summed over $\lambda\log^7 w$ timesteps. Thus, by a Chernoff bound, with probability at least $1 - 1/w^{\Theta(\lambda)}$, at least one job in S has a success. □

We are now ready to bound the number of anarchists having a particular window size.

Lemma 18. *Consider any window size w and any time interval $[t, t + w]$. Suppose that the instance is γ slack feasible for sufficiently small constant γ. With probability at least $1 - 1/w^{\Theta(\lambda)}$, the number of distinct jobs with window size w that are anarchists at any time during $[t, t + w]$ is at most $4w/\log^3 w$.*

Proof. We consider four separate time intervals of length $w/2$: $[t - w, t - w/2]$, $[t - w/2, t]$, $[t, t + w/2]$, and $[t + w/2, w]$. Observe that only the jobs released during one of these intervals can be anarchists during the time $[t, t + w]$.

Consider any one of those intervals I. Suppose at least $w/\log^3 w$ jobs with window w are released during interval I. Our goal is to bound the number of jobs released during I that can be anarchists. Let S be a set of the $w/\log^3 w$ jobs with earliest release time in this interval (ties can be broken arbitrarily). By Lemma 17, at least one job in S has a successful leader election with high probability. Because the deadlines of these jobs are far enough away, all other jobs released later in I follow that leader. Thus, with high probability, at most $w\log^3 w$ jobs released during I are anarchists. In the case that fewer jobs are released, this claim is vacuous. Summing across all four intervals completes the proof. □

The preceding lemma bounds the number of anarchists of a particular size. What we want is to be able to argue that anarchists have a good probability of success, i.e., that there are enough slots during their windows that have low contention. This is captured by the following lemma.

Lemma 19. *Let $w = 2^\ell$ be a window size, consider any time interval $I = [t, t + w]$ and suppose that the instance is γ slack feasible for sufficiently small constant γ. Then with probability at least $1 - 1/w^{\Theta(\lambda)}$, there exist at least $w/2$ anarchy slots in I that have contention at most $1/2$.*

Proof. By Lemma 18, we know that with high probability in w, there are not many anarchists with window sizes at least say $\sqrt{w}$ at any point during the interval. Their total contention in each slot is thus at most $\sum_{i=\ell/2}^{\infty}(4\cdot 2^i/i^3)(\lambda i/2^i) = 4\lambda\sum_{i=\ell/2}^{\infty} 1/i^2 \ll 8\lambda/\ell$. As long as ℓ is large enough (i.e., for small enough γ), this resolves to at most $1/4$ contention per slot.

It remains to bound the contention of anarchists with small windows. This proof follows the same structure as Lemma 11. Let w_0 denote the smallest window size and let 2^j, for $\lg(w_0) \leq j < \ell/2$ be a small window size. Divide I into subintervals of size 2^j. Our goal is to count the *total* contention generated by jobs with window 2^j. The total is the number of anarchists multiplied by the number of slots in the subinterval and the probability of transmitting in a slot. In any interval with few anarchists, the total contention is thus at most $O((2^j/j^3)(2^j)(\lambda j/2^j)) = O(\lambda 2^j/j^2)$. Summing across all low-anarchy subintervals, the total from jobs of window 2^j is at most $O(\lambda\,|I|\,/j^2)$. In any interval with too many anarchists, the total contention is at most $O((\gamma 2^j)(2^j)(\lambda j/2^j)) = O(\lambda j 2_j)$. The question is how many of these many-anarchist subintervals there can be. With sufficiently large λ, the probability of any subinterval having too many anarchists is at most $1/j^3$ by Lemma 18. Thus, by a Chernoff bound across at least $\sqrt{w}$ subintervals, the total contention from intervals with too many anarchists is at most $O(\lambda\,|I|\,/j^2)$ with high probability in w. Finally, summing across all j for sufficiently large w_0 gives a total contention of at most $|I|\,/8$. It follows that at most $|I|\,/2$ slots can have contention at least $1/4$ from small-window jobs. Adding the contention from large-window jobs gives a total contention of at most $1/2$ in at least half the slots. □

Corollary 20. *Consider any γ slack feasible instance for sufficiently small constant γ, and let j be any job that becomes an anarchist. Then with probability at least $1 - 1/w_j^{\Theta(\lambda)}$, job j successfully transmits its data message.*

Proof. By Lemma 19, at least half the anarchy slots during j's window have contention at most $1/2$. Thus, with high probability, at least half the slots have no other message transmission. In each of those otherwise silent slots, j transmits with probability $\lambda\log(w_j)/w_j$. The probability that j never transmits in an otherwise silent slot is thus at most $(1 - \lambda\log(w_j)/w_j)^{w_j} \leq 1/e^{\Theta(\lambda\log w_j)}$. □

5 RELATED WORK

Here we discuss some closely related work on backoff protocols and message transmission with deadlines/priorities.

The contention-resolution problem has been extensively studied from many different angles. Many prior works address a setting

where all devices/packets start at the same time—sometimes referred to as the *static case*—each with a message to send on the multiple-access channel. When the goal is to minimize the time until the first message is transmitted, several results are known [69, 76, 88]. A more general case is where k out of n devices need to succeed [8, 51].

A well-studied performance metric in the literature is makespan, which provides a measure of the time until all packets succeed [13, 14, 20, 28, 38, 47–49, 53, 53, 69, 72, 80, 87]. Binary exponential backoff and other natural variants where window sizes monotonically increase do not achieve the asymptotically optimal makespan [13]. However, a non-monotone algorithm called ***sawtooth*** is asymptotically optimal [8, 45, 52]. For packets with heterogeneous sizes, the performance of windowed algorithms has been studied in [14].

In modern wireless networks, several aspects make the contention-resolution problem challenging. With small devices, the on-board power supply is limited, and energy efficiency is addressed in several results [17, 29, 59]. Malicious interference, typically referred to as *jamming*, is treated in [3–5, 9, 17, 79, 82–85], while the impact on performance due to signal-propagation effects is investigated in in [41]. The impact of using multiple channels is considered in [42].

A natural extension is to consider when packets can arrive over time, perhaps in a worst-case fashion. In this setting, many results on contention resolution consider a statistical queuing-theory model with a focus on what packet-arrival rates are stable (see [47–49, 53, 53, 80]). In contrast, worst-case performance on a multiple-access channel is examined in the context adversarial queuing theory [6, 13, 34, 35]. A related notion is that of *saturated throughput*; this is, roughly, the maximum throughput when each player always has a packet to be sent [20, 87].

There are also several results related to contention resolution that address dynamic access to a shared resource. In [22], online packet arrivals are examined under a queuing model where performance is measured according to notions of queue-size competitiveness. In [7], the worst-case online arrivals of clients (rather than packets), each with data to upload, is considered. Finally, a problem of dynamic mutual exclusion, where subsets of processes must contend for access to a critical section at arbitrary times, is analyzed in [23].

The performance of adaptive algorithms—where packets employ channel feedback—and the power of randomized versus deterministic approaches has been investigated in several works. In [68], the authors present a deterministic, non-adaptive contention-resolution algorithm when players have access to a global clock. In [38], both adaptive and non-adaptive algorithms are considered without a global clock. Additionally, deterministic contention-resolution algorithms (which are non-adaptive) are considered in [63, 70]

The closely-related wake-up problem addresses how long it takes for a collection of devices to receive a wake-up transmission [31–33, 36, 60]. Finally, several works make use of a procedure for estimating the system size in order to speed up the performance of backoff algorithms [17, 21, 26, 27, 50].

While messages with deadlines have not been considered in the context of backoff, it is a natural question that crops up in many different domains. Many industrial and cyberphysical systems require real-time constraints on message deliveries for the proper operation of the system and real-time network protocols take these constraints into consideration [2, 40, 43]. Most of the algorithms designed in this context use centrally planned communication schedules that guarantee that no deadlines can be missed [25, 44, 73, 74, 86].

Message prioritization is also tied to traffic prioritization in the context of quality of service (QoS), which refers to a broad measurement of service performance, subsuming several aspects, such as delay, jitter, packet drop rate, availability, and transmission delay [24, 65, 67]. The prioritization of network traffic is a common ingredient in methods for providing QoS [56]. For example, precedence can be given to time-sensitive applications, such as voice-over-IP and multimedia streaming, where low-latency transport of data is required. Additionally, in systems where capacity is scarce, prioritization allows for the allotment of this resource to critical traffic during times of network congestion. We note that the categorization and methodology for assigning priority to different types of network traffic is a related but separate problem from the one treated here [61, 62, 90]. Various mechanisms for assigning priority exist, such as differentiated services in IPv4 [78], traffic classes in IPv6 [39], and enhancements to WiFi [30, 66, 77].

REFERENCES

[1] IEEE standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks – Specific requirements - Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pages 1–3534, 2016.

[2] R. Mangharam A. Rowe and R. Rajkumar. RT-Link: A time-synchronized link protocol for energy constrained multi-hop wireless networks. In *Third IEEE International Conference on Sensors, Mesh and Ad Hoc Communications and Networks (IEEE SECON)*, 2006.

[3] B. A. Aldawsari, B. S. Chlebus, and D. R. Kowalski. Broadcasting on adversarial multiple access channels. In *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, pages 1–4, Sep. 2019.

[4] Lakshmi Anantharamu, Bogdan S. Chlebus, Dariusz R. Kowalski, and Mariusz A. Rokicki. Medium access control for adversarial channels with jamming. In *Proceedings of the* 18^{th} *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 89–100, 2011.

[5] Lakshmi Anantharamu, Bogdan S. Chlebus, Dariusz R. Kowalski, and Mariusz A. Rokicki. Packet latency of deterministic broadcasting in adversarial multiple access channels. *J. Comput. Syst. Sci.*, 99:27–52, 2019.

[6] Lakshmi Anantharamu, Bogdan S. Chlebus, and Mariusz A. Rokicki. Adversarial multiple access channel with individual injection rates. In *Proceedings of the 13th International Conference on Principles of Distributed Systems (OPODIS)*, pages 174–188, 2009.

[7] Antonio Fernández Anta, Dariusz R. Kowalski, Miguel A. Mosteiro, and Prudence W. H. Wong. Scheduling dynamic parallel workload of mobile devices with access guarantees. *ACM Trans. Parallel Comput.*, 5(2), December 2018.

[8] Antonio Fernández Anta, Miguel A. Mosteiro, and Jorge Ramón Muñoz. Unbounded contention resolution in multiple-access channels. *Algorithmica*, 67(3):295–314, 2013.

[9] Baruch Awerbuch, Andrea Richa, and Christian Scheideler. A jamming-resistant MAC protocol for single-hop wireless networks. In *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 45–54, 2008.

[10] Naama Ben-David and Guy E. Blelloch. Analyzing contention and backoff in asynchronous shared memory. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 53–62, 2017.

[11] Michael A. Bender, Martin Farach-Colton, Sándor P. Fekete, Jeremy T. Fineman, and Seth Gilbert. Reallocation problems in scheduling. In *Proc. 25th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 271–279, Montreal, Canada, July 2013.

[12] Michael A. Bender, Martín Farach-Colton, Sándor P. Fekete, Jeremy T. Fineman, and Seth Gilbert. Reallocation problems in scheduling. *Algorithmica*, 73:389–409, June 2015.

[13] Michael A. Bender, Martin Farach-Colton, Simai He, Bradley C. Kuszmaul, and Charles E. Leiserson. Adversarial contention resolution for simple channels. In *Proc. 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 325–332, 2005.

[14] Michael A. Bender, Jeremy T. Fineman, and Seth Gilbert. Contention resolution with heterogeneous job sizes. In *Algorithms - ESA 2006, 14th Annual European*

Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings, pages 112–123, 2006.

[15] Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Maxwell Young. Scaling exponential backoff: Constant throughput, polylogarithmic channel-access attempts, and robustness. *J. ACM*, 66(1):6:1–6:33, 2019.

[16] Michael A. Bender, Tsvi Kopelowitz, William Kuszmaul, and Seth Pettie. Contention resolution without collision detection. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC)*, 2020.

[17] Michael A. Bender, Tsvi Kopelowitz, Seth Pettie, and Maxwell Young. Contention resolution with log-logstar channel accesses. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 499–508, 2016.

[18] Michael A. Bender, Tsvi Kopelowitz, Seth Pettie, and Maxwell Young. Contention resolution with constant throughput and log-logstar channel accesses. *SIAM J. Comput.*, 47(5):1735–1754, 2018.

[19] D. J. Bernstein. qmail — an email message transfer agent. http://cr.yp.to/qmail.html, June 1998.

[20] Giuseppe Bianchi. Performance analysis of the IEEE 802.11 distributed coordination function. *IEEE Journal on Selected Areas in Communications*, 18(3):535–547, September 2006.

[21] Giuseppe Bianchi and Ilenia Tinnirello. Kalman filter estimation of the number of competing terminals in an IEEE 802.11 network. In *Proceedings of the 22^{nd} Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 2, pages 844–852, 2003.

[22] Marcin Bienkowski, Tomasz Jurdzinski, Miroslaw Korzeniowski, and Dariusz R. Kowalski. Distributed online and stochastic queueing on a multiple access channel. *ACM Trans. Algorithms*, 14(2), May 2018.

[23] Marcin Bienkowski, Marek Klonowski, Miroslaw Korzeniowski, and Dariusz R Kowalski. Randomized mutual exclusion on a multiple access channel. *Distributed Computing*, 29(5):341–359, 2016.

[24] M. S. Borella, D. Swider, S. Uludag, and G. B. Brewster. Internet packet loss: measurement and implications for end-to-end QoS. In *Proceedings of the 1998 ICPP Workshop on Architectural and OS Support for Multimedia Applications Flexible Communication Systems. Wireless Networks and Mobile Computing (Cat. No.98EX206)*, pages 3–12, Aug 1998.

[25] A. Burns, J. Harbin, L. Indrusiak, I. Bate, R. Davis, and D. Griffin. Airtight: A resilient wireless communication protocol for mixed-criticality systems. In *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 65–75, Aug 2018.

[26] Frederico Cali, Marco Conti, and Enrico Gregori. Dynamic tuning of the IEEE 802.11 protocol to achieve a theoretical throughput limit. *IEEE/ACM Transactions on Networking*, 8(6):785–799, 2000.

[27] Frederico Cali, Marco Conti, and Enrico Gregori. IEEE 802.11 protocol: Design and performance evaluation of an adaptive backoff mechanism. *IEEE Journal on Selected Areas in Communications*, 18(9):1774–1786, 2000.

[28] Yi-Jun Chang, Wenyu Jin, and Seth Pettie. Simple contention resolution via multiplicative weight updates. In *Proceedings of the Second Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, pages 16:1–16:16, 2019.

[29] Yi-Jun Chang, Tsvi Kopelowitz, Seth Pettie, Ruosong Wang, and Wei Zhan. Exponential separations in the energy complexity of leader election. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 771–783, 2017.

[30] E. Charfi, L. Chaari, and L. Kamoun. PHY/MAC enhancements and QoS mechanisms for very high throughput WLANs: A survey. *IEEE Communications Surveys Tutorials*, 15(4):1714–1735, 2013.

[31] Bogdan S. Chlebus, Gianluca De Marco, and Dariusz R. Kowalski. Scalable wake-up of multi-channel single-hop radio networks. *Theoretical Computer Science*, 615(C):23 – 44, February 2016.

[32] Bogdan S. Chlebus, Leszek Gasieniec, Dariusz R. Kowalski, and Tomasz Radzik. On the wake-up problem in radio networks. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 347–359, 2005.

[33] Bogdan S. Chlebus and Dariusz R. Kowalski. A better wake-up in radio networks. In *Proceedings of 23rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 266–274, 2004.

[34] Bogdan S. Chlebus, Dariusz R. Kowalski, and Mariusz A. Rokicki. Adversarial queuing on the multiple-access channel. In *Proc. Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 92–101, 2006.

[35] Bogdan S. Chlebus, Dariusz R. Kowalski, and Mariusz A. Rokicki. Adversarial queuing on the multiple access channel. *ACM Transactions on Algorithms*, 8(1):5, 2012.

[36] Marek Chrobak, Leszek Gasieniec, and Dariusz R. Kowalski. The wake-up problem in multihop radio networks. *SIAM Journal on Computing*, 36(5):1453–1471, 2007.

[37] Bryan Costales and Eric Allman. *Sendmail*. O'Reilly, third edition, December 2002.

[38] Gianluca De Marco and Grzegorz Stachowiak. Asynchronous shared channel. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '17, pages 391–400, 2017.

[39] Steve Deering and Robert Hinden. Internet protocol, version 6 (IPv6) specification, 1998.

[40] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 73–84, April 2011.

[41] Jeremy T. Fineman, Seth Gilbert, Fabian Kuhn, and Calvin Newport. Contention resolution on a fading channel. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 155–164, 2016.

[42] Jeremy T. Fineman, Calvin Newport, and Tonghe Wang. Contention resolution on multiple channels with collision detection. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 175–184, 2016.

[43] HART Communications Foundation. IEC 62591: Industrial networks - Wireless communication network and communication profiles - wirelessHART (tm). Technical report, 2016.

[44] Chryssis Georgiou, Seth Gilbert, and Dariusz R. Kowalski. Meeting the deadline: On the complexity of fault-tolerant continuous gossip. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC âĂŹ10, page 247âĂŞ256, New York, NY, USA, 2010. Association for Computing Machinery.

[45] Mihály Geréb-Graus and Thanasis Tsantilas. Efficient optical communication in parallel computers. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 41–48, 1992.

[46] Leslie Ann Goldberg. Notes on contention resolution. 2000.

[47] Leslie Ann Goldberg and Philip D. MacKenzie. Analysis of practical backoff protocols for contention resolution with multiple servers. *Journal of Computer and System Sciences*, 58(1):232 – 258, 1999.

[48] Leslie Ann Goldberg, Philip D. Mackenzie, Mike Paterson, and Aravind Srinivasan. Contention resolution with constant expected delay. *Journal of the ACM*, 47(6):1048–1096, 2000.

[49] Jonathan Goodman, Albert G. Greenberg, Neal Madras, and Peter March. Stability of binary exponential backoff. *Journal of the ACM*, 35(3):579–602, July 1988.

[50] Albert G. Greenberg, Philippe Flajolet, and Richard E. Ladner. Estimating the multiplicities of conflicts to speed their resolution in multiple access channels. *Journal of the ACM*, 34(2):289–325, April 1987.

[51] Albert G. Greenberg and Shmuel Winograd. A lower bound on the time needed in the worst case to resolve conflicts deterministically in multiple access channels. *JACM*, 32(3):589–596, July 1985.

[52] Ronald I. Greenberg and Charles E. Leiserson. Randomized routing on fat-trees. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 241–249, 1985.

[53] Johan Hastad, Tom Leighton, and Brian Rogoff. Analysis of backoff protocols for multiple access channels. *SIAM Journal on Computing*, 25(4):1996, 740-774.

[54] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: Architectural support for lock-free data structures. In *Proceedings of the 20th International Conference on Computer Architecture*, pages 289–300, 1993.

[55] C. Hua and R. Zheng. Starvation modeling and identification in dense 802.11 wireless community networks. In *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, pages 1022–1030, April 2008.

[56] A. Iera, A. Molinaro, G. Ruggeri, and D. Tripodi. Improving QoS and throughput in single-and multihop WLANs through dynamic traffic prioritization. *IEEE Network*, 19(4):35–44, July 2005.

[57] V. Jacobson. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.*, 18(4):314–329, August 1988.

[58] Jangeun Jun and M. L. Sichitiu. Fairness and QoS in multihop wireless networks. In *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No.03CH37484)*, volume 5, pages 2936–2940 Vol.5, Oct 2003.

[59] Tomasz Jurdziński, Mirosław Kutyłowski, and Jan Zatopiański. Energy-efficient size approximation of radio networks with no collision detection. In *Proceedings of the 8th Annual International Conference (COCOON)*, pages 279–289, 2002.

[60] Tomasz Jurdzinski and Grzegorz Stachowiak. The cost of synchronizing multiple-access channels. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 421–430, 2015.

[61] Murat Karakus and Arjan Durresi. Quality of service (QoS) in software defined networking (SDN): A survey. *Journal of Network and Computer Applications*, 80:200 – 218, 2017.

[62] D. Kaur, K. Kaur, and V. Arora. QoS in WLAN using IEEE 802.11e: Survey of QoS in MAC layer protocols. In *2012 Second International Conference on Advanced Computing Communication Technologies*, pages 468–473, Jan 2012.

[63] Dariusz R. Kowalski. On selection problem in radio networks. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing, PODC 2005, Las Vegas, NV, USA, July 17-20, 2005*, pages 158–166, 2005.

[64] James F. Kurose and Keith Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[65] Li Zheng, Liren Zhang, and Dong Xu. Characteristics of network delay and delay jitter and its effect on voice over IP (VoIP). In *ICC 2001. IEEE International Conference on Communications. Conference Record (Cat. No.01CH37240)*, volume 1, pages 122–126 vol.1, June 2001.

[66] Stefan Mangold, Sunghyun Choi, Guido R. Hiertz, Ole Klein, and Bernhard Walke. Analysis of IEEE 802.11e for QoS support in wireless LANs. *IEEE Wireless Commun.*, 10(6):40–50, 2003.

[67] Y. Mansour and B. Patt-Shamir. Jitter control in QoS networks. *IEEE/ACM Transactions on Networking*, 9(4):492–502, 2001.

[68] Gianluca De Marco and Dariusz R. Kowalski. Fast nonadaptive deterministic algorithm for conflict resolution in a dynamic multiple-access channel. *SIAM Journal on Computing*, 44(3):868–888, 2015.

[69] Gianluca De Marco and Dariusz R. Kowalski. Contention resolution in a non-synchronized multiple access channel. *Theoretical Computer Science*, 689:1 – 13, 2017.

[70] Gianluca De Marco, Dariusz R. Kowalski, and Grzegorz Stachowiak. Deterministic contention resolution on a shared channel. In *39th IEEE International Conference on Distributed Computing Systems, ICDCS*, pages 472–482, 2019.

[71] A. Margolis, R. Vijayakumar, and S. Roy. Modelling throughput and starvation in 802.11 wireless networks with multiple flows. In *IEEE GLOBECOM 2007 - IEEE Global Telecommunications Conference*, pages 5123–5127, Nov 2007.

[72] Robert M. Metcalfe and David R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, July 1976.

[73] V. P. Modekurthy, D. Ismail, M. Rahman, and A. Saifullah. A utilization-based approach for schedulability analysis in wireless control systems. In *2018 IEEE International Conference on Industrial Internet (ICII)*, pages 49–58, Oct 2018.

[74] V. P. Modekurthy, A. Saifullah, and S. Madria. DistributedHART: A distributed real-time scheduling system for wirelessHART networks. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 216–227, April 2019.

[75] Amit Mondal and Aleksandar Kuzmanovic. Removing exponential backoff from TCP. *SIGCOMM Comput. Commun. Rev.*, 38(5):17–28, September 2008.

[76] K. Nakano and S. Olariu. Uniform leader election protocols for radio networks. *IEEE Transactions on Parallel and Distributed Systems*, 13(5):516–526, May 2002.

[77] Qiang Ni, Lamia Romdhani, and Thierry Turletti. A survey of QoS enhancements for IEEE 802.11 wireless LAN: Research articles. *Wireless Communications and Mobile Computing*, 4(5):547–566, August 2004.

[78] Kathleen Nichols, Steven Blake, Fred Baker, and David Black. RFC 2474: Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers, 1998.

[79] Adrian Ogierman, Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. Sade: competitive MAC under adversarial SINR. *Distributed Computing*, 31(3):241–254, Jun 2018.

[80] Prabhakar Raghavan and Eli Upfal. Stochastic contention resolution with short delays. *SIAM Journal on Computing*, 28(2):709–719, April 1999.

[81] Ravi Rajwar and James R. Goodman. Speculative lock elision: Enabling highly concurrent multithreaded execution. In *Proc. of the 34th Annual Intnl. Symposium on Microarchitecture*, pages 294–305, Austin, Texas, December 2001.

[82] Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. A jamming-resistant MAC protocol for multi-hop wireless networks. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 179–193, 2010.

[83] Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. Competitive and fair medium access despite reactive jamming. In *Proceedings of the* 31^{st} *International Conference on Distributed Computing Systems (ICDCS)*, pages 507–516, 2011.

[84] Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. Competitive and fair throughput for co-existing networks under adversarial interference. In *Proceedings of the* 31^{st} *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 291–300, 2012.

[85] Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. An efficient and fair MAC protocol robust to reactive interference. *IEEE/ACM Transactions on Networking*, 21(1):760–771, 2013.

[86] A. Saifullah, Y. Xu, C. Lu, and Y. Chen. Real-time scheduling for wirelessHART networks. In *2010 31st IEEE Real-Time Systems Symposium*, pages 150–159, Nov 2010.

[87] Nah-Oak Song, Byung-Jae Kwak, and Leonard E. Miller. On the stability of exponential backoff. *Journal of Research of the National Institute of Standards and Technology*, 108(4), 2003.

[88] Dan E. Willard. Log-logarithmic selection resolution protocols in a multiple access channel. *SIAM J. Comput.*, 15(2):468–477, May 1986.

[89] Yang Xiao. Performance analysis of priority schemes for IEEE 802.11 and IEEE 802.11e wireless LANs. *Wireless Communications, IEEE Transactions on*, 4(4):1506–1515, July 2005.

[90] M Aykut Yigitel, Ozlem Durmaz Incel, and Cem Ersoy. QoS-aware MAC protocols for wireless sensor networks: A survey. *Computer Networks*, 55(8):1982–2004, 2011.

[91] Qian M Zhou, Aiden Calvert, and Maxwell Young. Singletons for simpletons: Revisiting windowed backoff using chernoff bounds. In *Proceedings of the Tenth International Conference on Fun With Algorithms (FUN)*, 2020.