Scaling Exponential Backoff: Constant Throughput, Polylogarithmic Channel-Access Attempts, and Robustness

MICHAEL A. BENDER, Stony Brook University
JEREMY T. FINEMAN, Georgetown University
SETH GILBERT, National University of Singapore
MAXWELL YOUNG, Mississippi State University

Randomized exponential backoff is a widely deployed technique for coordinating access to a shared resource. A good backoff protocol should, arguably, satisfy three natural properties: (1) it should provide constant throughput, wasting as little time as possible; (2) it should require few failed access attempts, minimizing the amount of wasted effort; and (3) it should be robust, continuing to work efficiently even if some of the access attempts fail for spurious reasons. Unfortunately, exponential backoff has some well-known limitations in two of these areas: it can suffer subconstant throughput under bursty traffic, and it is not robust to adversarial disruption.

The goal of this article is to "fix" exponential backoff by making it scalable, particularly focusing on the case where processes arrive in an online, worst-case fashion. We present a relatively simple backoff protocol, Re-Backoff, that has, at its heart, a version of exponential backoff. It guarantees expected constant throughput with dynamic process arrivals and requires only an expected polylogarithmic number of access attempts per process.

Re-Backoff is also robust to periods where the shared resource is unavailable for a period of time. If it is unavailable for D time slots, Re-Backoff provides the following guarantees. For n packets, the expected number of access attempts for successfully sending a packet is $O(\log^2(n + D))$. For the case of an infinite number of packets, we provide a similar result in terms of the maximum number of processes that are ever in the system concurrently.

CCS Concepts: • **Theory of computation → Distributed algorithms**;

Additional Key Words and Phrases: Exponential backoff, contention resolution, distributed computing, algorithms, wireless networks, jamming attacks, robustness, throughput, adversarial scheduling

An earlier version of this work was presented at the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) (Bender et al. 2016a). This work was supported in part by National Science Foundation (NSF) grants CCF 1763680, CCF 1716252, CCF 1725543, CNS 1755615, CCF 1617618, CCF 1114809, CCF 1217708, CCF 1218188, CCF 1314633, IIS 1247726, IIS 1251137, CNS 1408695, CCF 1439084, CNS-1318294, CCF-1613772, and CNS-1816076. This work was also supported in part by AcRF Tier 1 grant T1 251RES1719, by Sandia National Laboratories, by NetApp, and by a research gift from C Spire.
Authors' addresses: M. A. Bender, Department of Computer Science, Engineering Dr., Office 245, Stony Brook University, Stony Brook, NY, 11794-2424, USA; email: bender@cs.stonybrook.edu; J. T. Fineman, Department of Computer Science, St. Mary's Hall, 3700 O St. NW Office 346, Georgetown University, Washington, DC, 20057-1232, USA; email: jfineman@cs.georgetown.edu; S. Gilbert, Department of Computer Science, Computing 1, 13 Computing Drive Office COM2-3-23, National University of Singapore, Singapore; email: seth.gilbert@comp.nus.edu.sg; M. Young, Department of Computer Science and Engineering, Butler Hall, 4665 George Perry Street, Office 312, Mississippi State University, MS, 39762 USA; email: myoung@cse.msstate.edu.

0004-5411/2018/12-ART6 $15.00
https://doi.org/10.1145/3276769

ACM Reference format:
Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Maxwell Young. 2018. Scaling Exponential Backoff: Constant Throughput, Polylogarithmic Channel-Access Attempts, and Robustness. *J. ACM* 66, 1, Article 6 (December 2018), 33 pages.
https://doi.org/10.1145/3276769

1 INTRODUCTION

Randomized exponential backoff (Metcalfe and Boggs 1976) is used throughout computer science to coordinate access to a shared resource. This mechanism applies when there are multiple processes (or devices, players, transactions, or packets) attempting to access a single, shared resource, but only one process can hold the resource at a time. Randomized backoff is implemented in a broad range of applications including local area networks (Metcalfe and Boggs 1976), wireless networks (Kurose and Ross 2002; Xiao 2005), transactional memory (Herlihy and Moss 1993), lock acquisition (Rajwar and Goodman 2001), email retransmission (Bernstein 1998; Costales and Allman 2002), and congestion control (e.g., TCP) (Mondal and Kuzmanovic 2008; Jacobson 1988).[1]

In randomized exponential backoff, when a process needs the resource, it repeatedly attempts to grab it. If two processes ***collide***—i.e., they try to grab the resource at the same time—then the access fails, and each process waits for a randomly chosen amount of time before retrying. After each collision, a process's expected waiting time doubles, resulting in reduced contention and a greater probability of success.

Given the prevalence (and elegance) of exponential backoff, it should not be surprising that myriad papers have studied the theoretical performance of randomized backoff. Many of these papers make queuing-theory assumptions on the arrival of processes needing the resource (Goldberg et al. 2000; Hastad et al. 1996; Raghavan and Upfal 1995, 1999; Capetanakis 1979). Others assume that all processes arrive in a batch (Goldberg et al. 1993; Geréb-Graus and Tsantilas 1992; Greenberg et al. 1987; Greenberg and Winograd 1985; Willard 1986) or adversarially (Bender et al. 2005; Chlebus et al. 2012, 2006). What may be surprising is how many foundational questions about randomized backoff remain unanswered, or only partially answered:

- *Throughput:* It is well known that classical exponential backoff achieves subconstant throughput, in the worst case. Is it possible for an exponential backoff variant to achieve constant throughput, particularly in dynamic settings, where arbitrarily large bursts of processes may arrive in any time slot, leading to varying resource contention over time?
- *Number of attempts:* On average, how many attempts does a process make before it successfully acquires the resource? Can one modify exponential backoff to achieve constant throughput, while still ensuring a few unsuccessful attempts? These questions make sense in applications where each access attempt has a cost. For example, in a wired network, an unsuccessful transmission wastes bandwidth. In a wireless network, an unsuccessful transmission wastes energy. In transactional memory, a transaction rollback (i.e., an unsuccessful attempt) wastes CPU cycles.
- *Robustness:* How robust is exponential backoff when the acquisition process suffers failures? An access attempt could fail *even when there is no collision*. Faults could arise due to hardware failures, software bugs, or malicious behavior. For example, a shared link may suffer from thermal noise, or a server may crash. A wireless channel may come under attack from jamming (see Xu et al. (2005) and Walters et al. (2007)), or a server may become unavailable

[1]Randomized backoff has also found use in a variety of cloud computing scenarios (Google 2014; Platform 2011; Services 2012).

When a process needs the resource:

— Set the window size $W = 2$.
— Repeat until the resource is acquired:
  ◦ Choose a slot t in the window uniformly at random. Try to acquire the resource at slot t.
  ◦ If the acquisition failed, then: (i) wait until the end of the current window, and (ii) set $W = 2W$.

Fig. 1. Exponential backoff.

due to a denial-of-service attack (DoS) (Hussain et al. 2003). In transactional memory, failures may result from best-effort hardware, since existing hardware implementations do not guarantee success even when there are no collisions (conflicts) between transactions.

1.1 Goal: Make Exponential Backoff Scalable

Randomized exponential backoff is "broken" in the worst case: it lacks good throughput guarantees (see, e.g., Bender et al. (2005)) and is not robust to failures. While randomized exponential backoff permits a relatively small number of access attempts, it suffers from $O(1/\log n)$ throughput when all n packets arrive as a single batch; furthermore, throughput can be much worse for other arrival patterns.

The goal of this article is to "fix" randomized exponential backoff to achieve good *asymptotic* performance. We modify the protocol as little as possible to maintain its simplicity, while finding a variant that (1) delivers constant throughput, (2) requires few access attempts, and (3) works robustly.

Since exponential backoff has many applications, there are many choices of terminology. Here we call the shared resource the ***channel*** and attempts to acquire the resource ***broadcasts***.[2] We subsume channel acquisition failures under an adversary, and a slot where the channel acquisition fails due to adversarial interference is said to be ***disrupted***.

We note that care is required in defining throughput, particularly in the presence of an adversary who arbitrarily schedules the arrival of packets and can disrupt the channel at will. Informally, time on the channel that is used due to a successful broadcast *or* disruption counts toward throughput. Conversely, time that is ***wasted*** due to the absence of any broadcast or the presence of a collision does not count toward throughput. Our formal definition is deferred until Section 2.2.

1.2 Related Work

A preliminary version of this work appeared in Bender et al. (2016a). Here, we have provided additional proof details, an expanded description of prior relevant work, and an extended discussion of open problems.

Exponential backoff is commonly studied in a network setting, where packets arriving over time are transmitted on a ***multiple-access channel***. This is a popular model and we compare and contrast our current work with prior results.

Queuing Models. For many years, most backoff analyses assumed models based on statistical queueing theory. In those cases, the focus was on finding stable arrival rates for packets (see Hastad et al. (1996), Goodman et al. (1988), Goldberg and MacKenzie (1996), Goldberg et al. (2000), and Aldous (1987)).

[2]We permit some slight abuses of the English language when packets may seem to broadcast themselves.

A related and important notion is that of ***saturated throughput,*** which is, roughly, the maximum throughput when each player always has a packet to be sent; this has been examined in Bianchi (2006) and Song et al. (2003).

Worst-Case/Adversarial Arrivals. When all the packets arrive at the same time, efficient protocols for a shared channel exist (Anta et al. 2013; Bender et al. 2006; Greenberg and Winograd 1985). Work has also been done in the context of adversarial queueing theory (Bender et al. 2005; Chlebus et al. 2012; Anantharamu et al. 2009).

When packets begin at different times, the problem is harder. The case of dynamic arrivals has been explicitly studied in the context of the "wake-up problem" (Chlebus and Kowalski 2004; Chlebus et al. 2005; Chrobak et al. 2007; Chlebus et al. 2016; Marco and Kowalski 2017; Jurdzinski and Stachowiak 2015), which looks at how long it takes for a single transmission to succeed when packets arrive dynamically. In contrast, our article focuses on achieving good bounds for a stream of packet arrivals (no fixed stations), when *all* packets must be transmitted.

In Marco and Kowalski (2015), players are awakened at *adversarially scheduled times* and have access to a global clock. However, players cannot detect when a collision occurs, though the channel does report whether a transmission was successful or not. A similar problem is considered in De Marco and Stachowiak (2017), but without a global clock.

Dynamic arrivals are considered in Bender et al. (2016b), which achieves constant throughput and $O(\log \log^* n)$ access attempts in expectation. However, this result is extremely sensitive to interference on the communication channel and therefore lacks any robustness guarantees.

The number of communication rounds required for contention resolution has been examined under different channel assumptions. Multiple channels with collision detection can speed up contention resolution (Fineman et al. 2016), even under the case of dynamic arrivals. Similarly, when the channel is governed by the signal-to-interference-and-noise ratio (SINR) equation, a nearly tight upper bound on the number of rounds is known (Fineman et al. 2016).

Robustness to Wireless Interference. As the focus on backoff protocols has shifted to include wireless networks, there has been an increasing emphasis on coping with noise and interference known as *jamming*. In a surprising breakthrough, the results in Awerbuch et al. (2008) showed that good throughput is possible with a small number of access attempts, even if jamming causes disruption for a constant fraction of the execution. A number of elegant results have followed Richa et al. (2010, 2011, 2012), Ogierman et al. (2018), Richa et al. (2013a, 2013b), and Klonowski and Pajak (2015), with good guarantees on throughput and access attempts.

Most of these jamming-resistant protocols do not assume fully dynamic packet arrivals. By this, we mean that these protocols are designed for the setting in which there are a fixed number of "stations" that are continually transmitting packets. In contrast, we are interested in the fully dynamic setting, where sometimes there are arbitrarily large bursts of packets arriving (lots of channel contention) and other times there are lulls with small handfuls of packets (little channel contention).

Also relevant are the results of Anantharamu et al. (2011), which concern the latency of deterministic algorithms for packet delivery on a multiple access channel under dynamic arrivals. In particular, the authors consider a (ρ, λ, b)-adversary who can inject $\rho|\tau| + b$ packets and jam at most $\lambda|\tau| + b$ slots within any contiguous segment τ consisting of $|\tau|$ slots, where $\rho + \lambda < 1$. In contrast, we are concerned with a model where the number of packets injected is not bounded by an injection rate less than 1, and our solution makes use of randomness.

First Successful Transmission/Estimating System Size. In Willard (1986), a shared-channel setting is considered where the goal is to minimize the *first* round in which some player succeeds in transmitting a packet. For N players starting at the same time, $\Theta(\log \log N)$ rounds (in expectation) are proved to be optimal.

In much of the literature, the number of players is (initially) unknown, yet this knowledge is often helpful in resolving contention on a shared channel. Consequently, various size estimation techniques have been developed in the context of exponential backoff and variants (Greenberg et al. 1987; Cali et al. 2000a, 2000b; Bianchi and Tinnirello 2003).

Relationship to Balanced Allocations. Scalable backoff is closely related to balls-and-bins games (Berenbrink et al. 2013, 2006; Richa et al. 2001; Vöcking 2003; Azar et al. 1999; Mitzenmacher 2001; Berenbrink et al. 2012; Cole et al. 1998). Bins correspond to time slots and balls correspond to packets. The objective is for each ball to land in its own bin; if several balls share the same bin, they are rethrown. The flow of time gets modeled by restrictions on when balls get thrown and where they may land. The results in our article are ultimately about scalable randomized algorithms and asymptotic analysis for dealing with bursts robustly and scalably.

2 MODEL: CONTENTION RESOLUTION ON A MULTIPLE-ACCESS CHANNEL

Time is discretized into **slots** where a process can broadcast a packet, i.e., access the shared resource. We do not assume a global clock; i.e., there is no universal numbering scheme for slots.

When there is no transmission (or adversarial disruption, see below) during a slot, we call that slot ***empty***. A slot is ***full*** when one or more packets are broadcast in that slot. When exactly one packet is broadcast in a slot, that packet transmits successfully, and the full slot is ***successful***. When two or more packets are broadcast during the same slot, a ***collision*** occurs in this (full) slot. When there is a collision, there is "noise" on the channel; all packets transmitting are unsuccessful.

We assume that a device transmitting a packet can determine whether its transmission is successful; this is a standard assumption in the backoff literature (for examples, see Kwak et al. (2005), and Goldberg et al. (2000)), unlike the wireless setting where a full medium access control (MAC) protocol would address acknowledgments and other issues.

We also assume that a device that is listening on the channel, but not transmitting, can tell whether a given slot is full or empty. We do not require the listener to distinguish between collisions and successful transmissions.

For simplicity of presentation, we assume there are actually two channels that processes can use simultaneously: a "control channel" and a "data channel." We explain in Section 7 how to implement our solution using only one channel.

2.1 Arbitrary Dynamic Packet Arrivals

New packets arrive arbitrarily over time. We do not assume any bounded arrival rate. A packet is ***live*** at any time between its arrival and its successful transmission. The number of packets in the system may vary arbitrarily over time, and this number is unknown to the packets. Without loss of generality, we assume that throughout the execution of the protocol, there is at least one live packet in every slot. (If not, simply ignore any slot during which there are no live packets.)

We postulate an ***adversary***, who governs two aspects of the system's dynamics: (1) The adversary determines the (finite) number of new packets that arrive in each slot. (2) The adversary may arbitrarily ***disrupt*** slots. A disruption appears as a full slot to all packets; any packet transmitted simultaneously fails. This model corresponds to collisions on Ethernet or a 1-uniform adversary in wireless networks (see Richa et al. (2010)).

The adversary is adaptive with one exception—the adversary decides a priori whether the execution contains infinitely many packets or a finite number n of packets. For the finite case, the adversary chooses n a priori. The packets themselves do not know whether the instance is infinite or finite (meaning that they cannot know n). In all other ways, the adversary is adaptive: it may make all arrival and disruption decisions with full knowledge of the current and past system state; at the end of a given slot, the adversary learns everything that has happened in that slot.

2.2 Throughput and Waste

We define throughput in the natural way: for an interval I, the throughput $\lambda \in [0, 1]$ is the fraction of successful slots in the interval I. Recall that for the purposes of throughput and waste, we only consider slots when there is at least one packet live in the system.

We also define a notion of "waste." A slot is ***wasted*** if there was a missed opportunity for a successful transmission: the slot was empty or more than one packet was broadcast. Otherwise, the slot is ***nonwasted*** , i.e., successful or disrupted. A disrupted slot is not seen as wasted, since it could never be used for a successful packet transmission. The ***nonwaste*** $\Lambda \in [0, 1]$ of an interval I is the fraction of nonwasted slots in I, and the ***waste*** is $1 - \Lambda$. In the absence of disruption, throughput and nonwaste are identical.

Definition 2.1. Consider interval $\mathcal{I}$ having $N_{\mathcal{I}}$ successful transmissions and $D_{\mathcal{I}}$ disrupted slots. The ***throughput*** of $\mathcal{I}$ is $\lambda = N_{\mathcal{I}}/|\mathcal{I}|$, the ***nonwaste*** is $\Lambda = (N_{\mathcal{I}} + D_{\mathcal{I}})/|\mathcal{I}|$, and the ***waste*** is $1 - \Lambda$. The throughput/waste of a finite execution is the throughput/waste for the interval $[0, T]$, where T is the latest any packet completes.

Definition 2.2. An infinite instance has Λ-nonwaste if, for any slot t, there exists a slot $t' \geq t$ where interval $[0, t']$ has Λ nonwaste. An infinite instance has λ-throughput if, for any slot t, there exists a slot $t' \geq t$ where interval $[0, t']$ has λ throughput.

The throughput/nonwaste does not depend on the arrival rate, even with no restrictions on arrivals. The arrival rate could be higher than feasible for an arbitrary period of time (e.g., two packets arrive every slot), and the system continues to deliver good throughput (even as the number of backlogged packets necessarily grows).

There are also no restrictions on the distribution of disruptions. The adversary can choose to disrupt arbitrarily large intervals of slots. When there are enough nondisrupted slots, constant throughput resumes.

2.3 Results

We devise a "(R)obust (E)fficient" backoff protocol, Re-Backoff, that (1) delivers constant throughput, (2) guarantees few failed access attempts, and (3) works robustly. We assume no global broadcast schedule, shared secrets, or centralized control. Let D be the total number of slots disrupted by the adversary.

Theorem 2.3. *For a finite number of packets n injected into the system, where n is fixed a priori but not revealed,* Re-Backoff *guarantees at most an expected constant fraction of wasted slots and spends $O(\log^2(n + D))$ access attempts per packet, in expectation.*

Each packet injected into the system may be viewed as corresponding to a unique process that is attempting to send this packet. Theorem 2.3 implies that Re-Backoff delivers constant throughput when at least a constant fraction strictly less than 1 of the slots are not disrupted.

Corollary 2.4. *There exists a constant c such that if $D \leq cn$, then* Re-Backoff *achieves expected constant throughput. A stronger property holds:* Re-Backoff *attains an expected makespan (completion time of last packet) of $O(n)$.*

In fact, the metric of expected makespan is both stronger and more desirable than expected throughput. To see why, observe that throughput is defined as the fraction of successful slots; see Section 2.2. Consider a scenario in which the throughput is $1/2$ with probability $1/2$, and 0 with probability $1/2$: then the expected throughput is constant, but the expected makespan is infinite. Really, we want the expected *reciprocal* of throughput to be $O(1)$, which is equivalent to saying that

RE-BACKOFF for a packet u that has been active for s_u slots.

— With probability $\frac{c \max\{\ln s_u,\, 1\}}{s_u}$, send busy tone on the control channel.
— With probability $\frac{d}{s_u}$, send on the data channel and, if successful, then terminate.
— Monitor the data channel. If at least $\lceil \gamma s_u \rceil$ data slots have been empty ($\gamma = 15/16$) since node u became active, then become inactive.

RE-BACKOFF for an inactive packet

— Monitor each control slot. If a slot is empty, then become active in the next slot.

Fig. 2. Pseudocode for RE-BACKOFF.

the expected makespan is $O(n)$. In this article, we focus on minimizing the expected makespan, optimizing the expected throughput as a byproduct.

An implication of Theorem 2.3 is that the number of access attempts is small relative to the number of disrupted slots. Specifically, (1) our protocol is parsimonious with broadcast attempts in the absence of disruption, and (2) if a packet has been in the system for T slots, then it makes expected $O(\log^2 T)$ access attempts, regardless of how many of these T slots were adversarially blocked.

Extending these results to the infinite case, we show that in an infinite execution, for a countably infinite number of slots, the protocol achieves both good throughput and few access attempts.

For any time t, denote by D_t the number of slots disrupted before t, and by η_t the maximum number of packets concurrently in the system before t.

THEOREM 2.5. *Suppose an infinite number of packets get injected into the system. Then for any time s, there exists a time $t \geq s$ such that at time t,* RE-BACKOFF *has at most constant waste with probability 1 and expected average $O(\log^2(\eta_t + D_t))$ access attempts per packet.*

Again, this implies that RE-BACKOFF yields constant throughput in infinite executions:

COROLLARY 2.6. *There exists a constant c such that, for every time s there exists a time $t \geq s$ where if $D_t \leq ct$, then there is constant throughput until time t.*

3 ALGORITHM

This section presents our backoff protocol. To simplify the presentation, we assume throughout that there are two communication channels: a ***data channel***, on which packets are broadcast, and a ***control channel***, on which a "busy signal" is broadcast. See Section 7 for how to implement this algorithm on one channel.

For a given packet u, let s_u be the number of slots it has been active for. Our protocol has the following structure (see Figure 2 for pseudocode):

— Initially, each packet is ***inactive***; it makes no attempt to broadcast on either channel.
— Inactive packets monitor the control channel. As soon as the packet observes an empty slot on the control channel, it becomes ***active***.
— In every time slot, an active packet broadcasts on the data channel with probability proportional to how long it has been active; i.e., packet u broadcasts with probability d/s_u, for a constant $d = 1/2$. It also broadcasts on the control channel with probability $c \max(\ln s_u, 1)/s_u$, for a constant $c > 0$.

— A packet remains active until it transmits successfully or sees too many empty slots. Specifically, if packet u has observed $\lceil \gamma s_u \rceil$ empty slots, where $\gamma = 15/16$, then the packet reverts to an ***inactive*** state, and the process repeats.

In essence, our protocol wraps exponential backoff with a coordination mechanism (i.e., the busy channel) to limit entry, and with an abort mechanism to prevent overshooting. In between, it runs something akin to classical exponential backoff (instantiated by broadcasting in round t with probability $1/t$, instead of using windows). One aspect that we find interesting is how little it takes to fix exponential backoff.

4 PROTOCOL DESIGN

This section gives the intuition behind the design of Re-Backoff.

Consider the following simple protocol that Re-Backoff builds upon. Packets are initially inactive. Whenever there are no active packets, all packets in the system become active and run an asymptotically optimal ***batch*** backoff protocol on the data channel (e.g., SawTooth Backoff (Bender et al. 2005)). Active packets *all* broadcast a ***busy tone*** on every control-channel slot, and inactive packets wait for silence on the control channel.[3] The busy tone contains *no data*, and it serves only to prevent newcomers from activating until all currently active packets have transmitted successfully.

The busy tone yields a ***batch invariant***: there is only one batch running in the system at a time, which allows the throughput guarantees of the batch protocol to extend to arbitrary arrivals. Unfortunately, this basic protocol yields an unacceptable number of access attempts—one attempt per active packet per slot due to the busy tone. But even this primordial protocol is interesting because it shows a simple strategy for achieving constant throughput, in contrast to classical exponential backoff (we elaborate below).

We require a cheaper busy tone. A natural approach is for active packets to broadcast randomly on the control channel. This modified protocol broadcasts less, but it suffers the occasional ***control failure***, where the busy tone disappears even though some packets are still active.

The question is how active packets should respond to control failures. A plausible approach is to reset *every* packet, making every packet in the system restart in a single new batch. With no disruptions, this new protocol achieves constant throughput with a polylogarithmic number of access attempts.

But it is not robust to disruptions. The adversary has too much control: it can spoof the busy tone until packets have backed off a lot. The adversary then stops, and now packets have a very low probability of making an access attempt before a control failure causes a reset, which forces packets to join a new batch. Using this strategy, the adversary can prevent almost all of the packets in each batch from broadcasting successfully, forcing them to reset many times. Specifically, the adversary can keep packets in the system for $T \gg n$ slots, and it can force $T^{\Theta(1)}$ access attempts rather than polylog$(n + T)$ access attempts, as specified by Theorems 2.3 and 2.5.

Re-Backoff addresses the previous concern by avoiding immediate resets; a packet only resets once a constant fraction of slots during its current batch are empty. Intuitively, the reset condition means that any packet that was reset could easily have chosen one of the empty slots and just got unlucky. Any packet that enters a batch has at least a constant probability of broadcasting successfully in that batch and at most a constant probability of resetting. Therefore, in Re-Backoff, a packet joins an expected constant number of batches before succeeding.

[3]Busy tones are also used in mutual exclusion and MAC protocols (see, e.g., Haas and Deng (2002) and Shong Wu and Li (1988)) for coping with hidden terminal effects.

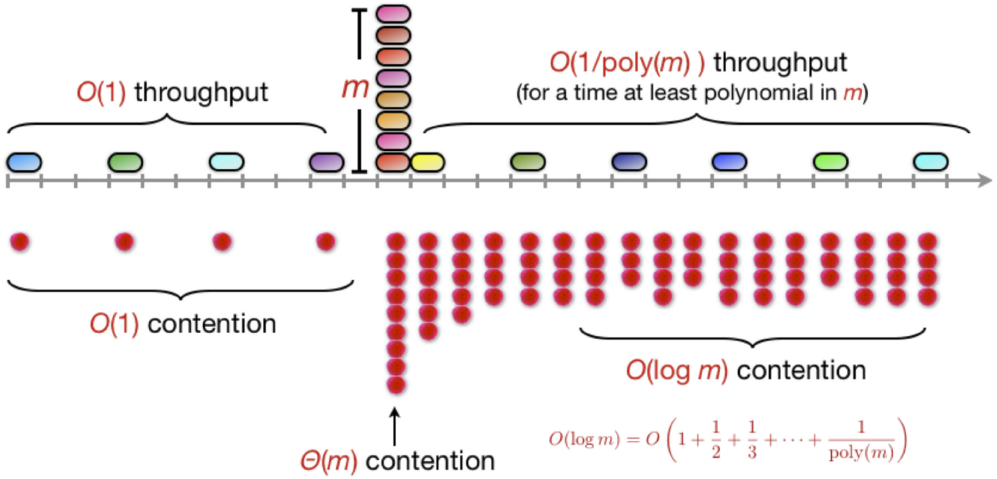

Fig. 3. Illustration of how exponential backoff struggles with batches (Bender et al. 2005). Shown above the line are packet arrivals, and below the line are transmission attempts. When there is more than one red dot below the line, there is a collision.

And so Re-Backoff sacrifices the batch invariant; multiple batches may exist in the system simultaneously and, consequently, we have put the throughput guarantee in jeopardy. This is because batch protocols do not perform well with dynamic arrivals. Even exponential backoff, which is already suboptimal on batches, performs asymptotically worse under dynamic arrivals.

This suboptimal behavior is illustrated in Figure 3 where the packet arrivals illustrated make up a steady stream of one new packet every three time steps, plus a single batch of m packets. Initially, the throughput is fine. After the batch arrives, the contention grows large; even after $\Theta(m/\log m)$ time steps subsequent to when the batch arrives, w.h.p. the contention from this batch is $\Omega(\log m)$, and so the probability that any packet succeeds is $O(1/\text{poly}(m))$. Over these $\Theta(m/\log m)$ steps, the steady stream of packets accumulates, and the ith packet contributes $\Theta(1/i)$ to the contention. Therefore, by the sum of a harmonic series, the contention from the steady stream of packets is logarithmic in m, and will continue to be so for $\Omega(\text{poly}(m))$ steps into the future.

In the design of Re-Backoff, the probabilistic busy tone and delayed reset serve together as a "leaky mutual exclusion" protocol, which keeps out many overlapping batches but allows others to "leak" into the critical section. This contrasts with the (error-free) busy tone and aggressive reset mechanisms, each of which deterministically ensures mutual exclusion.

Most of the technical contribution of our article has to do with proving that despite leaky mutual exclusion, Re-Backoff still guarantees constant contention (sum of broadcast probabilities) a constant fraction of the time, and therefore ensures constant nonwaste (and constant throughput when at most a constant fraction of slots are jammed). The idea is to prove that there are enough prefixes of slots so that if the contention is much more or much less than a constant for X slots, then there are $\Omega(X)$ slots where contention is $\Theta(1)$.

Digging deeper, the technical hurdle that contention arguments seem to have is that contention changes over time in ways that are hard to characterize. For example, if the contention in a given time slot comes from a small number of young packets, then it will drop quickly over time (unless another batch activates), whereas if the contention comes from a large number of older packets, then it will drop gradually. Thus, there is a funny and unpredictable way in which the contention changes as a function of time.

Besides its complexity, what makes this proof unusual to us is that we are deprived of some of our favorite tools: high-probability arguments, e.g., using Chernoff bounds. This tool is denied to us because the bursts may be arbitrarily smaller than the number of packets ever to enter the system.

Ultimately, we have a rather simple protocol that maintains, at its core, exponential backoff—while at the same time delivering the three desirable properties: constant throughput, few attempts, and robustness.

5 THROUGHPUT AND WASTE ANALYSIS

In this section, we analyze Re-Backoff, showing that it achieves at most constant waste in both the finite and infinite cases.

Let s_i^t be the age of packet i in slot t, i.e., the number of slots that it has been active. At time t, we define the ***contention*** to be $X(t) = \sum_i 1/s_i^t$, where we sum over all the active packets. Thus, the expected number of broadcasts on the data channel in slot t is $dX(t)$.

For every slot t, we define the value σ_t to be the minimum age out of all active packets such that the following hold: (1) $\sum_{i:0<s_i^t\le\sigma_t} 1/s_i^t \ge X(t)/2$ and (2) $\sum_{i:s_i^t\ge\sigma_t} 1/s_i^t \ge X(t)/2$. That is, active packets with age $\le \sigma_t$ have at least half the contention, and active packets with age $\ge \sigma_t$ have at least half the contention. We call these two sets the ***young*** and ***old*** packets, respectively. Note that packets with age exactly σ_t are *both* young and old.

We say that a ***control failure*** occurs in slot t if no packet broadcasts on the control channel during the slot. Recall that (1) a packet activation can occur only immediately after a control failure and (2) a packet j ***resets*** at time t if t is the first slot during j's lifetime $[t - s_j^t, t]$ of s_j slots, for which at least γs_j slots are empty.

Overview. In Section 5.1, we relate performance to contention. The tricky part is to bound *how often* and *for how long* the contention stays high. In Section 5.2, we break the execution up into epochs, structuring the changes in contention. We can then analyze the control failures (Section 5.3) and resets (Section 5.4) as a function of contention. This leads to a key result (Corollary 5.13) in Section 5.5 that shows that an epoch is "good" (in some sense) with constant probability.

One tricky aspect remains: the adversary may use the results from earlier epochs to bias later epochs by injecting new packets at just the wrong time. We introduce a simple probabilistic game, *the bad borrower game*, to capture this behavior and show that it cannot cause much harm (in Sections 5.6 to 5.8). Finally, we assemble the pieces in Sections 5.9 and 5.10, showing that we achieve at most a constant-factor waste.

5.1 Individual Slot Calculations

The next two lemmas look at the probability of a broadcast as a function of the contention, first looking at successful broadcasts and then all broadcasts—even those that result in a collision. We note that the proofs use $d = 1/2$, specifically relying on the inequality $d \le 1/2$. This is not an absolute requirement on d, but the analysis is simplified by using the inequality $1 - x \ge e^{-2x}$, which holds for any nonnegative quantity $x \le 1/2$ (where, in our analysis, x is related to d).

Lemma 5.1. *For a given slot t in which there is no disruption, the probability that some packet successfully broadcasts at time t is at least $\frac{dX(t)}{e^{2dX(t)}}$.*

Proof. Packet j is successful with probability $\frac{d}{s_j}\prod_{i\ne j}(1-\frac{d}{s_i})$. At most one packet is successful, so the success events for each packet are disjoint. The probability that some packet succeeds is thus at least $\frac{d}{s_1}\prod_i(1-\frac{d}{s_i}) + \frac{d}{s_2}\prod_i(1-\frac{d}{s_i}) + \cdots + \frac{d}{s_k}\prod_i(1-\frac{d}{s_i}) = (\prod_i(1-\frac{d}{s_i}))\cdot\sum_j \frac{d}{s_j} \ge \frac{dX(t)}{e^{2dX(t)}}$.

The denominator follows from the fact that $1 - x \geq e^{-2x}$ for $0 \leq x \leq 1/2$, and hence $\prod_i (1 - \frac{d}{s_i}) \geq e^{-2d \sum_i (1/s_i)}$. □

Lemma 5.2. *The probability that some packet is broadcast (not necessarily successfully) in slot t is at least $1 - e^{-d\,X(t)}$ and at most $1 - e^{-2d\,X(t)}$. The probability of a collision in the slot is at most $(1 - e^{-2d\,X(t)})^2$.*

Proof. The probability that no packets broadcast is $\prod_i (1 - \frac{d}{s_i}) \leq e^{-d\,X(t)}$. Conversely, $\prod_i (1 - \frac{d}{s_i}) \geq e^{-2d\,X(t)}$ by the fact that $1 - x \geq e^{-2x}$ for $0 \leq x \leq 1/2$. Let p be the probability that a slot is full. Then, the probability of a collision in that slot is at most p^2. Since $p \leq 1 - e^{-2d\,X(t)}$, the claim follows. □

5.2 Epochs, Streaks, and Interstitial Slots

An execution is divided into two types of periods, ***epochs*** and ***interstitial slots***, and we describe these below.

Definition 5.3. Each time t_0 when a packet is activated, a new ***epoch*** begins (and any earlier epoch ends). To describe the duration of an epoch, we have two cases:

- *Case 1.* If the contention is not too high at the start of an epoch, specifically, if $X(t_0) < 8$, then the epoch consists of a single slot t_0. We call such an epoch a ***unit epoch***.
- *Case 2.* Else, $X(t_0) \geq 8$, and then the epoch is subdivided further into a sequence of ***streaks***, with the first streak beginning at t_0. If a streak begins at time t, then it ends at slot $t' = t + \sigma_t$ (or at the start of a new epoch, whichever occurs first). If $X(t') < 8$, then the epoch ends. Otherwise, another streak begins at time t' and ends at time $t' + \sigma_{t'}$.

When an epoch ends—the criteria for when this happens is specified in both cases above—either a new epoch begins (if a new packet is activated) or there is a gap between epochs called the ***interstitial slots***. By definition, no packet activates during interstitial slots (otherwise a new epoch begins), and the contention is less than 8 (otherwise, a streak is still ongoing).

We elaborate on the difference between a unit epoch and an interstitial slot. An epoch, including a unit epoch, starts only when a packet activates. In contrast, an interstitial slot occurs only after an epoch (including a unit epoch) ends. Note that to have an interstitial slot, the epoch must end due to the contention falling below 8 after a sequence of streaks, and not because a new packet is activated.

Whenever we reference an epoch, unless we specify otherwise, this is meant to include a unit epoch.

We now bound the change in contention during a streak.

Lemma 5.4. *Assume that some streak begins at time t and that no control failures occur during the streak. Then $X(t + \sigma_t) \leq 3X(t)/4$.*

Proof. During the streak, all the young packets at time t *at least* double in age (since they each have age at most σ_t), so their contention reduces by at least half. Moreover, by definition, the young packets at time t have contention at least $X(t)/2$, so the total contention reduces by at least $X(t)/4$. Since there are no control failures, there are no new packets activated and hence no increase in contention. □

Lemma 5.5. *Assume that some streak begins at time t, where $X(t) \geq 8$, and that no resets occur during the streak. Then for all $t' \in [t, t + \sigma_t]$, $X(t') \geq X(t)/8$. Moreover, this $X(t)/8$ is a lower bound on the contention contributed by just the old packets.*

Proof. Since there are no resets (and no activations, by definition) during the streak, the contention only decreases due to packets completing and due to increasing age. Consider the old packets at time t (which have contention at least $X(t)/2$ at time t). Since each of these packets at most doubles in age (since they have age $\geq \sigma_t$), their total contention would remain at least $X(t)/4$ throughout the streak.

Some of these packets may finish, thus reducing the contention further. Assume that the old packet with the largest contention completes in every slot—notice that each such packet that finishes reduces the contention by at most $1/\sigma_t$. Thus, if one such packet finishes in each slot of the streak, the total contention is reduced by at most $\sigma_t/\sigma_t = 1$. Thus, throughout the streak, the total contention from old packets remains at least $X(t)/4 - 1 \geq X(t)/8$ (since $X(t) \geq 8$). □

Corollary 5.6. *Consider any slot t' that is part of an epoch. If no packets reset during the epoch before time t', then $X(t') \geq 1$.*

Proof. If t' is the first slot of the epoch, then $X(t') \geq 1$ trivially since the packet whose activation started the epoch contributes 1 to contention. Note that this argument applies to unit epochs.

Otherwise, there exists some time $t \leq t'$ that marks the start of the current streak. Since there have been no resets, Lemma 5.5 implies $X(t') \geq X(t)/8$. By definition of epochs/streak, $X(t) \geq 8$. We conclude that $X(t') \geq 1$. □

5.3 Control Failures

Next we look at the probability of a control failure as a function of contention. The next lemma argues that for the next slot in a streak (that has not yet had any failures), the old packets provide enough contention to make a control failure in the next slot unlikely. The subsequent lemma takes a union bound over all slots in the streak to conclude that it is unlikely for any control failure to occur in the streak.

Lemma 5.7. *For a fixed time t when a streak begins, consider a control slot at time t' during the streak. Assume that there are no control failures or resets during the streak prior to time t'. Then the probability of a control failure in slot t' is at most $(\sigma_t)^{-\frac{cX(t)}{4}}$.*

Proof. The probability of a control failure in slot t' is at most (see Figure 2):

$$\begin{aligned}\prod_i \left(1 - \frac{c \ln s_i^{t'}}{s_i^{t'}}\right) &\leq e^{-c\sum_i \frac{1}{s_i^{t'}} \ln s_i^{t'}} \\ &= \sigma_t^{-c\sum_i \frac{\ln s_i^{t'}}{s_i^{t'} \ln \sigma_t}} \\ &\leq \sigma_t^{-c\sum_{\text{old } i} \frac{1}{s_i^{t'}}} \\ &\leq \sigma_t^{-cX(t)/8}.\end{aligned}$$

The third line follows from the fact that $\ln s_i^{t'} \geq \ln \sigma_t$ for all old packets. The fourth line follows from Lemma 5.5. Note that the final result is a function of t, not t'. □

Lemma 5.8. *Consider a streak beginning at time t. Assume that no reset occurs during the streak. For any target $b > 0$, there exists a sufficiently large choice of a constant c such that the following holds: the probability that a control failure occurs in the interval $[t, t + \sigma_t]$ is at most $(\sigma_t)^{-bX(t)}$.*

Proof. Assuming there are no control failures during time $[t, t']$ for $t' \leq t + \sigma_t$, the probability of a control failure at time t' is at most $(\sigma_t)^{-\frac{cX(t)}{8}}$ by Lemma 5.7. Taking a union bound over the σ_t time slots, the probability of a control failure happening in any time slot is at most $(\sigma_t)^{-\frac{cX(t)}{8}+1}$. □

5.4 Bounding Resets

We next bound the probability that a reset takes place during an epoch. We show that with constant probability, a packet does not reset during an epoch; this is true since for any prefix of the epoch, there are sufficiently many broadcasts to prevent a reset.

The argument proceeds in the following manner. We begin by examining a sequence of independent Bernoulli trials, each with (at least) probability p of success. The intuition is that these trials correspond to slots in an epoch, and a success corresponds to a full slot, whereas a failure corresponds to an empty slot. Recall from the specification of the Re-Backoff that a packet resets if it observes $\lceil \gamma s_u \rceil$ empty slots, where $\gamma = 15/16$.

Lemma 5.9 establishes a lower bound on the probability that the first i Bernoulli trials (slots) contain at least $ip/4$ successes. Importantly, the probability bound established depends on p and d only; there is no dependence on i (we explain the importance of this below).

Next, in Lemma 5.10, we establish two important bounds. First, we obtain a lower bound on the value of p for use in Lemma 5.9. Second, we argue that a reset occurs only if there exists an i such that the first i slots of the epoch contain less than $ip/4$ successes. This allows us to apply Lemma 5.9 and obtain a bound on the probability that any packet will reset in a particular epoch, which depends only on d, and has no dependence on i. Therefore, this bound applies to any number of slots in an epoch, which simplifies our analysis when this lemma is applied later in Lemma 5.12.

Lemma 5.9. *Consider a sequence of independent Bernoulli trials each with probability at least p of success. The following holds with probability at least $(\frac{1}{2})\, p^{16/p}$: for all i, the first i trials contain at least $ip/4$ successes.*

Proof. Break up the trials into geometrically increasing subsequences of 2^k trials each. We say that a "failure" occurs in the k^{th} subsequence if there are fewer than $p2^k/2$ successful trials within that subsequence. Using a Chernoff bound, the failure probability is at most $e^{-p2^k/8}$. Using a union bound, the probability that a failure occurs in any subsequence $k \geq \lg(1/p) + 4$ is at most $\sum_{k=\lg(1/p)+4}^{\infty} e^{-p2^k/8} = \sum_{j=1}^{\infty} e^{-2^j} \leq 1/2$. Therefore, with probability at least $q \cdot (1/2)$, the first $16/p$ trials are a success *and* every subsequence has at least $p2^k/2$ successes.

Now, suppose there is no failure in any subsequence; i.e., each has at least $p2^k/2$ successful trials. Pick any cutoff point in the subsequence of size 2^i and examine the total of 2^{i+1} trials up to this point. The previous subsequences for $k = 1, \ldots, i-1$ each contain at least $p2^k/2$ successful trials, for a total of at least $p \cdot 2^i/2$ successful trials. Therefore, up to the cutoff point, at least a $p/4$-fraction of the trials are successful. □

We conclude in the next lemma that the probability that any packet resets during the epoch is low. Note that the following analysis depends on the constant d, and recall that $d = 1/2$ (given our discussion in Section 5.1).

Lemma 5.10. *The probability that* any *packet resets during a particular epoch is at most a positive constant whose value is strictly less than 1 and only depends on d.*

Proof. Consider an execution of the protocol. Let t be the start of the epoch in question, and suppose that no packets reset before time $t' \geq t$ belonging to the same epoch. Then by Corollary 5.6, we have $X(t') \geq 1$. Combining this fact with Lemma 5.2, we have that slot t' is empty with probability at most $e^{-dX(t')} \leq e^{-d}$; i.e., it is full with probability at least $1 - e^{-d}$.

We shall now map the execution to the game in Lemma 5.9, beginning from the start of the epoch at time t. In our sequence of Bernoulli trials, a slot $t' \geq t$ corresponds to a "success" if it is full and "failure" otherwise. The game continues until either a packet resets (we lose) or the epoch ends (we win).

We have already argued that until we lose, we have a success/full-slot probability of at least $p = 1 - e^{-d}$. We claim also (proof to follow) that we lose only if there exists an i such that the first i slots of the epoch contain less than $ip/4$ successes. Assuming the claim, we can apply Lemma 5.9 to conclude that we win with constant probability at least $(\frac{1}{2})\, p^{16/p} = (\frac{1}{2})(1 - e^{-d})^{16/(1-e^{-d})} > 0$.

To prove the claim, observe that no packets activate during the epoch, since activation starts a new epoch. Thus, any packets active at time t' must observe all slots in $[t, t']$. Let i be the number of slots in this interval. If the interval $[t, t']$ has at least $(1 - \gamma)i = i/16$ successes, then by the specification of the algorithm, no packets reset at time t'—note that a packet can only be below threshold at time t' if it was also below threshold at time t, which means it would have reset sooner. With $p = 1 - e^{-d} = 1 - e^{-1/2} > 1/4$, we have that a reset occurs only if there are less than $i/16 < ip/4$ successes, as claimed above. □

5.5 Successful Streaks

The notation $S(t)$ refers to a streak beginning at time t and continuing for σ_t slots. A streak is ***successful*** if there are no resets or packet activations during the streak. Notice that during a successful streak, we know that the contention is always at least $X(t)/8$ (by Lemma 5.5), and that at the end of the streak, it is at most $3X(t)/4$ (by Lemma 5.4). We say that successful streaks $S(t)$ and $S(t')$ are consecutive if $t' = t + \sigma_t$.

The following lemma says that the length of any given streak dominates the sum of the lengths of all previous streaks in the epoch.

Lemma 5.11. *Let $S(t_1), \ldots, S(t_k)$ be a set of consecutive successful streaks for which there is nonzero contention over each streak. Then, for $j = 2, \ldots, k$, $\sigma_{t_j} \geq \sum_{i=1}^{j-1} \sigma_{t_i}$.*

Proof. For any $j = 2, \ldots, k$, consider the set of active packets after $\sum_{i=1}^{j-1} \sigma_{t_i}$ slots. Since the contention is nonzero, there exist remaining active packets. Moreover, since there are no injections during successful streaks, each packet must have been active at time t_1. Thus, each packet has an age $\sum_{i=1}^{j-1} \sigma_{t_i}$ at the start of the streak $S(t_j)$.

Recall that by definition (see the beginning of Section 5), σ_{t_j} is the age of some active packet at time t_j. Since all packets have age at least $\sum_{i=1}^{j-1} \sigma_{t_i}$, it follows that σ_{t_j} must also be at least this large. □

We next prove a key result: an ***epoch is successful***, meaning that all streaks during the epoch are successful, with constant probability. To prove the result, we analyze the change in contention and use the union bound over the streaks.

Lemma 5.12. *For a nonunit epoch beginning at time t, with constant probability, every streak is successful.*

Proof. Throughout the proof, let $t_1 = t$. By assumption that the streak is nonunit, we have $X(t_1) \geq 8$. Let $S(t_1), \ldots, S(t_k)$ be consecutive streaks such that k is the first index in these consecutive streaks where the contention drops below 8. Define $\tau_j = \sum_{1 \leq i \leq j} \sigma_{t_i}$.

For the special case where we might have $\sigma_{t_1} = 1$, the probability of a control failure in that single slot is at most $e^{-cX(t_1)} \leq e^{-8c}$ since $X(t_1) \geq 8$. This probability can be made as small as we desire by setting the appropriate value of c. For all other streaks, $\sigma_{t_i} \geq 2$.

By Lemma 5.8, the probability of a control failure in $S(t_1)$, which spans the interval $[t_1, t_1 + \sigma_{t_1}]$, is at most $(\sigma_{t_1})^{-b_1 X(t_1)}$, for some constant $b_1 > 0$. Similarly, the probability of a control failure in $S(t_2)$, which spans the interval $[t_2, t_2 + \sigma_{t_2}] = [t_1 + \sigma_{t_1}, t_1 + \sigma_{t_1} + \sigma_{t_2}] = [t_1 + \tau_1, t_1 + \tau_2]$, is at most $(\sigma_{t_2})^{-b_2 X(t_2)} = (\sigma_{t_2})^{-b_2 X(t_1+\tau_1)} \leq (1/2)^{b_2 X(t_1+\tau_1)}$ since $\sigma_{t_2} \geq 2$, and for some constant $b_2 > 0$.

In general, the probability of a control failure over $S(i)$, which spans the interval $[t_i, t_i + \sigma_{t_i}] = [t_1 + \sigma_{t_1} + \cdots + \sigma_{t_{i-1}}, t_1 + \sigma_{t_1} + \cdots + \sigma_{t_i}] = [t_1 + \tau_{i-1}, t_1 + \tau_i]$, is at most $(\sigma_{t_i})^{-b_i X(t_i)} = (\sigma_{t_i})^{-b_i X(t_1+\tau_{i-1})} \leq (1/2)^{b_i X(t_1+\tau_{i-1})}$, since $\sigma_{t_i} \geq 2$, and for some constant $b_i > 0$.

Using this bound on the probability of a control failure in each streak, and letting $b \geq \max_i\{b_i\}$ be a sufficiently large constant, we obtain that the probability of a control failure over the interval $[t_1, t_1 + \tau_k]$—that is, control failure in any of the streaks—is at most

$$\max\left\{\left(\frac{1}{e}\right)^{c\cdot X(t_1)}, \left(\frac{1}{2}\right)^{b\cdot X(t_1)}\right\} + \left(\frac{1}{2}\right)^{b\cdot X(t_1+\tau_1)} + \cdots + \left(\frac{1}{2}\right)^{b\cdot X(t_1+\tau_{k-3})} + \left(\frac{1}{2}\right)^{b\cdot X(t_1+\tau_{k-2})} + \left(\frac{1}{2}\right)^{b\cdot X(t_1+\tau_{k-1})}.$$

By Lemma 5.4, the contention decreases by at least a multiplicative factor of 3/4 from one streak to the next. Additionally, the contention at the beginning of any streak is at least 8. Therefore, we observe that the contention decreases by more than an additive value of two when going from $S(i)$ to $S(i+1)$. For example, if the contention at the beginning of $S(i+1)$ was 8 (which is the minimum it can be), the contention at the beginning of the previous streak $S(i)$ must have been at least $(4/3)8$, and so the difference between these contention values is $(4/3)8 - 8 = 8/3 > 2$.

With this observation in hand, we return to the above sum. Note that the largest summand is the last one since this is when the contention is smallest. The summand to the left is at least a factor of 4 smaller, by our observation above that the contention is decreasing by at least 2. Similarly, the term to the left of that is at least another factor of 4 smaller, and so on. Therefore, the value of the sum is upper-bounded by a geometric series; this sum is no more than $2 \cdot (1/2)^{b\cdot X(t+\tau_{k-1})} \leq 2 \cdot (1/2)^{b\cdot 8}$, which we can set to any desired (small) constant δ depending only on a sufficiently large constant b, which can be achieved by setting c (recall that c is a constant used in the specification of Re-Backoff, as described in Section 3).

By Lemma 5.10, for any epoch, the probability that any packet resets is at most a positive constant strictly less than 1, which depends only on d; denote this by q'. Therefore, the epoch is successful with probability at least $1 - ((1 - q') + \delta) = q' - \delta \geq \varepsilon$ for some constant $\varepsilon > 0$ depending only on c and d. □

We define an epoch to be ***disrupted*** if at least 1/4 of the slots in the epoch are disrupted; otherwise, the epoch is ***nondisrupted***. The following corollary shows that we get constant throughput in a nondisrupted epoch with constant probability.

Corollary 5.13. *For a unit epoch that is nondisrupted, with constant positive probability a packet broadcasts. For a nonunit epoch with length T', with constant positive probability: (i) every streak in the epoch is successful; (ii) the last streak is of length at least $T'/2$; (iii) the contention throughout the last streak is between 1 and 8; and (iv) if the epoch is nondisrupted, then at least $\Omega(T')$ packets are broadcast.*

Proof. Conclusion (i) follows from Lemma 5.12; conclusion (ii) follows from Lemma 5.11. Conclusion (iii) follows because there are no resets or packet activations; hence, the contention decreases by at most a factor of 8 by Lemma 5.5. Conclusion (iv) follows from (ii) and (iii), and from observing that, in a nondisrupted slot in the last streak, there is a constant probability that a packet is broadcast, and a constant probability that one is not (due to an empty slot or a collision). By Lemma 5.11, if less than 1/4 of the epoch is disrupted, then less than half of the slots in the last streak are disrupted. Of the less than $T'/4$ nondisrupted slots in the last streak, in expectation, only a constant fraction are *not* successful broadcasts. Thus, by Markov's inequality, with constant probability, at most a constant fraction of these less than $T'/4$ slots are not successful broadcasts, and hence with constant probability, at least $\Omega(T')$ packets are broadcast. □

5.6 Bad Borrower Game

We have shown that each epoch is ***good*** (satisfying Corollary 5.13) with constant probability. We now abstract away details, defining a simple game between two players: the ***lender*** and the ***borrower***. There are two key parameters: a probability p and a fraction $\alpha \in (0, 1)$. The game proceeds in ***iterations***, where in each, the borrower borrows an arbitrary (adversarially chosen) amount of money from the lender, at least \$1. With probability p, at the end of the iteration, the borrower repays a fraction α of the money.

The correspondence to our situation is as follows: each iteration is associated with a nondisrupted epoch, the length of the epoch defines the money borrowed, and the number of successful broadcasts defines the money repaid. In a good epoch, which occurs with constant probability p, if the epoch is nondisrupted, then we get constant throughput and hence the borrower is repaid an α fraction of his or her money. In a bad epoch, by contrast, we allow for the worst case, which is no money paid back at all (no throughput, all waste).

For the ***finite bad borrower game***, there is a predetermined maximum amount that the lender can be repaid: after the borrower has repaid n dollars, the game ends. This corresponds to a finite adversary that injects exactly n packets.

In the ***infinite bad borrower game***, the game lasts forever, and an infinite amount of money is lent. This corresponds to infinite instances, where the adversary injects packets forever.

5.7 Finite Bad Borrower Game

Our goal in this section is to show that, when the borrower has repaid n dollars, he or she has borrowed at most $O(n)$ dollars. This corresponds to showing that n packets are successfully broadcast in $O(n)$ time, ignoring the interstitial slots (which we will come back to later).

We assume throughout this section that n is the maximum amount of money repaid throughout the game; i.e., the adversary injects n packets in an execution. A simple correspondence lemma bounds the amount of money that the borrower can borrow:

Lemma 5.14. *In every iteration of the finite bad borrower game, ≥ 1 dollar and $\leq n$ dollars are borrowed.*

Proof. The fact that the borrower borrows at least \$1 follows by definition. Assume the borrower borrows n dollars, i.e., that the associated epoch lasts for at least n slots. Recall that the final streak in the associated epoch has at least $n/2$ slots, and at the beginning of that final streak, the contention must have been at least 8 (or the epoch would have ended). Since the final streak has length at least $n/2$, there must be a set of old packets with age $\geq n/2$ that collectively have contention at least 4 (by the definition of old packets in Section 5). This implies there must be at least $2n$ such packets, which is impossible, given the bound of n packets total. Thus, it is impossible to have an epoch of length n, and hence to borrow more than n dollars in an iteration of the finite bad borrower game. □

We now argue, via an analysis of the expected repayments, that when the finite bad borrower game ends, the expected cost to the lender is $O(n)$:

Lemma 5.15. *Over an execution of the bad borrower game, the expected number of dollars borrowed is at most $n/(p\alpha) + n = O(n)$.*

Proof. We analyze the dollars repaid as follows: we assume that for every dollar lent, it is paid back with probability $p\alpha$. Note that these random choices are correlated: for a given iteration, either an α fraction of the dollars are repaid (with probability p) or no dollars are paid back (with probability $1 - p$). For a given iteration of the game, if there are k dollars borrowed, we see that the expected number of dollars repaid is $pk\alpha$.

What is the expected number of dollars we must lend in order for n dollars to be repaid? The answer is $n/(p\alpha)$; i.e., after $O(n)$ dollars have been borrowed, all n dollars have been repaid. In the last iteration, there can be at most n additional dollars lent (as part of the iteration where the last dollar is repaid), by Lemma 5.14, yielding an expected $n/(p\alpha) + n$ borrowed dollars. □

5.8 Infinite Bad Borrower Game

In order to analyze infinite executions, we look at the infinite bad borrower game. In this case, we show that there are an infinite number of times where the borrower has repaid at least a $p\alpha/3$ fraction of the total dollars borrowed.

Lemma 5.16. *For any iteration r of the infinite bad borrower game, with probability 1 there is a later iteration $r' > r$ such that if the lender has lent k dollars through iteration r', then the borrower has repaid at least $kp\alpha/3$ dollars.*

We can model the infinite bad borrower game as a one-dimensional biased random walk with variable step size. Let X_r be the value of the random walk in iteration r, where $X_0 = 0$. Our goal is to construct a random walk as $X_r = 0$ when exactly a $p\alpha/3$ fraction of borrowed money has been repaid, and $X_r > 0$ when more than enough has been repaid. To do so, we renormalize the random walk so that every dollar paid back moves a distance of $3/(p\alpha)$ to the right.

We thus define the random walk as follows. Suppose that x_r dollars are lent in iteration r, where x_r may vary from iteration to iteration. Then with probability p, the iteration is good (an α fraction of the borrowed money has been repaid) and $X_r = X_{r-1} - x_r + (x_r\alpha)(3/(p\alpha)) = X_{r-1} - x_r + 3x_r/p \geq X_{r-1} + 2x_r/p$. (The last inequality is used only to simplify the analysis.) With probability $(1-p)$, the iteration is bad and we have $X_r = X_{r-1} - x_r$.

Since $E[X_r] = X_{r-1} + x_r > 0$, it is reasonable to assume that this walk will tend rightward and cross the origin regardless of the negative starting point. The remainder of this subsection is devoted to a rigorous proof that such a random walk is indeed positive infinitely often, which is what is required to complete the proof of Lemma 5.16. (Note that we cannot say that the random walk eventually remains always positive after some point, which would be true of a biased random walk with fixed step size. The issue is that the adversary can always introduce very large steps, for example, by always borrowing more than X_{r-1} dollars in step r; this allows the adversary the possibility of moving leftward of the origin at any point.)

5.8.1 Analysis of Random Walk. We consider a biased random walk where the step size in each step is variable and chosen by the adversary. Before each step t, the adversary chooses the step size, and then the biased coin is flipped to determine whether the ***walker's*** step is to the right (if the result of the flip is heads) or to the left (if the result of the flip is tails). For ease of exposition, we use the terms "right" and "left" interchangeably with "heads" and "tails," respectively.

For the purposes of the following argument, we assume that $\Pr(\text{right}) = p \geq 1/5$ and $\Pr(\text{left}) = (1-p) \leq 4/5$. The use of concrete values makes our exposition simpler, while conveying the main points of the argument. Later, in Section 5.8.2, we explain how to generalize our argument for any constant bias in favor of moving right.

Lemma 5.17. *Consider a sequence of flips with a biased coin where* $\Pr[\text{heads}] \geq 1/5$ *and* $\Pr[\text{tails}] \leq 4/5$. *Then, there exists a constant $\hat{t}$ such that with probability at least $1 - e^{-\Theta(\hat{t})}$, for any step in the sequence $t \geq \hat{t}$, the number of tails is at most 9 times the number of heads.*

Proof. Consider a sequence of t coin flips. Let the random variable X_t denote the number of tails in the first t flips of the biased coin. Then, $E[X_t] \leq 4t/5$.

Let A_t denote the event that there are more than 9 times as many tails as there are heads in our sequence of t coin flips; that is, $X_t > 9t/10$. Using a standard Chernoff bound and plugging

$\delta = 1/8$, we obtain

$$\Pr[A_t] = \Pr[X_t > 9t/10] = \Pr[X_t > (1+\delta)(4t/5)] \leq e^{-(1/8)^2 E[X_t]/3} = e^{-t/240}.$$

We wish to bound the probability of the event $A = \bigcup_{j=0}^{\infty} A_{\hat{t}+j}$. By the union bound,

$$\Pr[A] \leq \sum_{t=\hat{t}}^{\infty} \Pr[A_t] = \sum_{t=\hat{t}}^{\infty} e^{-t/240}.$$

For $\hat{t}$ sufficiently large, this sum is strictly less than 1, and so $\Pr[A] \leq e^{-\Theta(\hat{t})}$. □

Terminology and Overview. The execution of the walk is broken into a sequence of ***phases***. Suppose phase h begins with the walker at ***distance*** Δ_h to the left of the origin. Then, phase h lasts for $\Theta(\Delta_h)$ steps, where the constant inside the Θ is derived later on (the last paragraph of the proof of Lemma 5.18). Conversely, if the walker begins at or to the right of the origin, then the phase lasts for one time step only, and the next step is the beginning of the next phase. For most of our argument below, we analyze a single phase; therefore, we drop the subscript denoting the phase index and use Δ to refer to Δ_h.

We will demonstrate a positive probability of success within a phase, which implies that there is some point in time where a constant fraction of the outstanding borrowed amount has been repaid (the walker has reached or exceeded the origin).

To be clear, phases are a conceptual component of our argument, and they do not correspond directly to an iteration. In a phase, the adversary may use multiple different step sizes; that is, a phase may capture what happens over multiple iterations, where each iteration r may use a different step size x_r.

For each phase, we divide all possible step sizes into $L+1$ ***size classes***, where $L = \lceil \log_{11/10} \Delta \rceil$; that is, the number of size classes depends on the walker's initial distance to the left of the origin at the beginning of the phase. The size class i, for $i = 1, \ldots, L$, contains all steps whose size is in the interval $[(11/10)^{i-1}, (11/10)^i)$. Thus, the steps in any size class have the same length to within a factor of 11/10; this value is chosen because it guarantees rightward movement later in our argument. By the setup of our random walk, for a fixed size class $i = 1, \ldots, L$, each flip of tails moves the walker to the left by at most $(11/10)^i$, while each flip of heads moves the walker to the right by at least $(2/p)(11/10)^{i-1}$. Finally, the largest size class, i.e., size class $i = L+1$, contains all steps that are larger than Δ.

In our proof, we model randomness as follows. First, we assume a separate random source of coin flips for each size class. Second, for each size class, we even allow the adversary to *know* the string of coin flips in advance; that is, the adversary can see the outcome of each flip in advance for each size class and use this information to select a size class that best suits a strategy for stymieing the walker. These two aspects of our argument do not weaken the adversary (indeed, the second can only help the adversary), and they strengthen our argument.

Our goal is to prove that in each phase there is at least a constant probability that the walker moves to the right of the origin in some step. If the walker moves to the right of the origin in some step during a phase, then we say that the walker ***succeeds*** in that phase; otherwise, the walker ***fails***. Since we have an infinite random walk, this is sufficient to establish Lemma 5.16, thus proving our claim about the infinite bad borrower game.

Lemma 5.18. *For each phase, the walker succeeds with at least a positive constant probability.*

Proof. There is a constant probability that the first $\Theta(\hat{t})$ coin flips in the largest size class are all heads, where $\hat{t}$ is a sufficiently large constant. That is, the first $\Theta(\hat{t})$ steps in the largest size class are to the right. If this is not true, we concede the phase to the adversary and move on to

the next phase. This concession is acceptable since, in order to establish Lemma 5.16, the walker needs only to succeed with constant probability greater than 0, and $\Theta(\hat{t})$ is a constant.

Given the above, we only analyze those phases where the first $\Theta(\hat{t})$ coin flips in the largest size class are heads (moving the walker rightward), and we highlight two details. First, we specify $\Theta(\hat{t})$ coin flips rather than a single coin flip from the largest size class to be heads. This is because a single heads may be insufficient to push the walker to the right of the origin if the adversary is first able to select coin flips from smaller size classes that move the walker farther to the left prior to using the largest size class. Recall that, for the purposes of our proof, the adversary is able to see the outcome of coin flips for each size class, and so this adversarial behavior is possible. However, a sufficiently large constant number of steps from the largest size class are sufficient, and this is why we analyze only phases that meet this condition.

Second, this means the adversary will not use this largest size class since the adversary sees that the first $\Theta(\hat{t})$ coin flips in the largest size class allow the walker to reach the right of the origin, and so the adversary must use steps from smaller size classes.

Throughout the remainder of this argument, we consider a phase where the walker starts at an arbitrary distance of Δ to the left of the origin.

Let $B_{j,t}$ denote the event that for size class j, for any time step t or greater, there are at most 9 times as many tails as heads. Let $C_t = B_{L,t} \cap B_{L-1,t+1} \cap B_{L-2,t+2} \cap \cdots \cap B_{1,t+L-1}$. By Lemma 5.17, $Pr[B_{L-k,t+k}] = 1 - e^{-\Theta(t+k)}$ for $k = 0, \ldots, L-1$ for a sufficiently large constant t. Therefore, for some sufficiently large constant t, we have $Pr[C_t] \geq 1 - e^{-\Theta(t)}$ by the sum of a geometric series.

Why do we choose at most 9 times the number of tails as heads? Recall that steps within the same size class i vary in size within $[(11/10)^{i-1}, (11/10)^i)$. Therefore, in the worst case, it is possible that all leftward steps could be the largest in that particular size class (denote this step size by ℓ) and that all rightward steps could be the smallest in that particular size class (denote this by s). If the number of tails is at most 9 times the number of heads, then leftward movement by the walker is less than $9\,\ell \leq 9\,(11/10)^i = (99/10)(11/10)^{i-1} < 10\,(11/10)^{i-1}$, and the rightward movement by the walker is at least $(2/p)s \geq (2/p)(11/10)^{i-1} = 10\,(11/10)^{i-1}$. Thus, the net movement by the walker is rightward *within a size class j* whenever event $B_{j,t}$ occurs.

Next, we assume that event $C_{\hat{t}}$ occurs in the phase; if not, then we concede the phase to the adversary and move onto the next phase.

We make the following two claims:

Subclaim 1. If event $C_{\hat{t}}$ occurs, then the farthest left that a random walk can go using just the first $1, \ldots, L$ size classes is $\sum_{i=0}^{L-1}(\Delta/(11/10)^i)(\hat{t}+i) = O(\Delta\,\hat{t})$.

The bound given in Subclaim 1 follows immediately in the (worst) case that all steps are to the left. This is useful since we know that the walker's distance to the left of the origin is bounded prior to the inevitable net rightward movement that event $C_{\hat{t}}$ signifies.

Subclaim 2. If event $C_{\hat{t}}$ occurs, then it does not help the adversary to use any steps from size class $L+1$ since the adversary has already lost by the time it gets even one leftward step in this size class.

Subclaim 2 is true because we are only analyzing phases where the first $\Theta(\hat{t})$ coin flips in size class $L+1$ are to the right (which occurs with a constant, strictly positive probability). In these phases of interest, the adversary cannot opt to use the largest size class.

For the final portion of our argument, we set the number of steps in the phase to be sufficiently large that no matter how the adversary chooses the steps from the size classes, the walker is pushed to the right of the origin, by Subclaim 1. To do so, for each size class, we count the maximum number of steps possible beyond which the walker is to the right of the origin. We are pessimistic; for a

fixed size class, we assume steps to the right are the smallest in that size class, and steps to the left are the largest in that size class. Despite this, we still have net rightward movement per size class.

We start with size class L. Recall that the step size lies in $[(11/10)^{L-1}, (11/10)^{L})$, where $L = \log_{11/10} \Delta$, and by Subclaim 1, the walker's maximum distance to the left of origin is $O(\Delta\, \hat{t})$ when $C_{\hat{t}}$ occurs. It follows that there cannot be more than $c\hat{t} = \Theta(\hat{t})$ steps from size class L for a sufficiently large positive constant c, since this would push the walker to the right of the origin. This is true no matter what the adversary does with any smaller size class.

Similarly, there cannot be more than $(11/10)c\hat{t}$ steps from size class $L-1$ because these steps would also push the walker to the right of the origin. In general, there cannot be more than $(11/10)^i c\hat{t}$ steps in size class $L-i$ because these steps would push the walker to the right of the origin.

Summing up the number of steps in all of the size classes at most L yields at most $(11/10)^{L+1} c\hat{t} = (11/10)\Delta c\hat{t}$.

We need not consider size class $L+1$ given Subclaim 2. Therefore, as long as the number of steps in a phase is greater than $(11/10)\Delta\, c\hat{t}$, then the following holds: with at least constant probability that is strictly positive, the adversary cannot prevent the walker from being pushed to the right of the origin since there are simply too many steps. □

5.8.2 Generalizing Our Argument. We describe how the argument given in Lemma 5.18 changes when the probability of a heads is smaller than 1/5. Suppose that $\Pr[\text{heads}] = 1 - \Pr[\text{tails}]$ is any small positive constant. Now Lemma 5.17 does not hold in its current form, but a virtually identical argument can be used to prove the more general claim:

Consider a sequence of flips with a biased coin where $\Pr[\textit{heads}] \geq p$ *and* $\Pr[\textit{tails}] \leq 1-p$ *for any (small) constant* $p > 0$. *Then, there exists a positive constant* $\hat{t}$ *such that with probability at least* $1 - e^{-\Theta(\hat{t})}$, *for any step in the sequence* $t \geq \hat{t}$, *the number of tails is at most* $R = (1+\varepsilon)(1-p)/p$ *times the number of heads for some arbitrarily small constant* $\varepsilon > 0$ *depending on sufficiently large* $\hat{t}$.

In our proof of Lemma 5.18, we argued that a bound of at most 9 times as many tails as heads is sufficient to show net rightward movement by the walker given that the ratio of largest to smallest step size within a size class is less than 11/10. In our more general case, these constants require adjustment.

In particular, we can set the ratio of the largest step size to the smallest step size within a fixed size class to be strictly less than $(R+2)/(R+1)$; our original argument above used $R = 9$, and so we used a ratio of 11/10. This bound on the ratio of step sizes within a size class alters the number of different size classes. A ratio of maximum to minimum step size of 11/10 is possible using $1 + \log_{11/10} \Delta$ size classes. More generally, using $1 + \log_{(R+2)/(R+1)} \Delta$ size classes suffices, where size class j contains steps in the range $[(R+2)/(R+1)^{j-1}, (R+2)/(R+1)^{j})$.

Now, the leftward movement by the walker is less than $R((R+2)/(R+1))^i$, and the rightward movement by the walker is at least $(2/p)((R+2)/(R+1))^{i-1}$. To have net rightward movement by the walker, we want

$$R\left(\frac{R+2}{R+1}\right)^{i} < (R+1)\left(\frac{R+2}{R+1}\right)^{i-1} \leq (2/p)\left(\frac{R+2}{R+1}\right)^{i-1},$$

which holds for $\varepsilon \leq 1/(1-p)$ (which we can set by considering a sufficiently large $\hat{t}$), since then $R+1 \leq (1+\varepsilon)(1-p)/p \leq 2/p$. Thus, the overall movement by the walker is rightward within a size class j whenever event $B_{j,t}$ occurs.

With these adjustments, the remainder of the argument provided above generalizes to handle the case of any small positive constant probability of heads.

5.8.3 Application to the Infinite Bad Borrower Game. We can now prove Lemma 5.16:

Proof. Recall from the beginning of Section 5.8 that the random walk of interest is described as follows. Let X_r be the value of the random walk in iteration r. Again, we have:

$$X_r = \begin{cases} X_{r-1} + \left(\frac{2}{p}\right) x_r, & \text{with probability } p \\ X_{r-1} - x_r, & \text{with probability } 1-p. \end{cases}$$

By Lemma 5.18 (and the generalization of this argument discussed in Section 5.8.2), in each phase with at least constant probability that is strictly positive, the walker will cross the origin moving rightward. Since this is true of each phase, there is always a later iteration where the lender is repaid a constant fraction of the outstanding borrowed amount. □

As a corollary, if we only consider the epochs, ignoring the contribution from the interstitial slots, we can show constant throughput for infinite executions. Specifically, we can use the infinite bad borrower game to define "measurement points," thus showing that in an infinite execution, there are an infinite number of points at which we get constant throughput (if we ignore the contribution from the interstitial slots).

Corollary 5.19. *If we take an infinite execution and remove all slots that are not part of an epoch, then the resulting execution has at most a constant fraction of waste.*

We later show (Section 5.10) that the contribution from the interstitial slots does not hurt the waste, meaning that we get at most a constant factor of waste taking into account all slots.

5.9 Interstitial Slots, Expected Waste, and Expected Throughput for Finite Instances

We begin by considering the finite case where there are n packets injected. We bound the length of the interstitial slots, after which we prove at most an expected constant factor of waste.

We first argue that for a packet's lifetime, any prefix of at least two slots is at least a constant fraction full.

Lemma 5.20. *Suppose that a packet u is active for the time interval $[t, s]$. Then for any time t' with $t < t' \leq s$, at least a $1 - \gamma = 1/16$ fraction of the slots in the interval $[t, t']$ are full.*

Proof. The proof is by contradiction. Suppose that $[t, t']$ is strictly less than a $(1-\gamma)$-fraction full. Let $x = t' + 1 - t$ be the total number of slots in the interval, and let k be the number of full slots in the interval. Then we have $x > \frac{1}{1-\gamma}k = 16k$. Since $x > 16k$ is an integer, $x \geq 16k + 1$. Thus, the subinterval $[t, t' - 1]$ contains at most k full slots across $x - 1 \geq 16k$ slots, meaning that a reset would occur at or before time $t' - 1$. □

Our next lemma extends the above argument to cover all interstitial slots. We would like to say that the first t time slots of the entire execution include at most a γ-fraction of empty slots. This is not necessarily true—the first slot could be empty. The issue is that Lemma 5.20 does not apply to the first slot of a packet's lifetime. But we can make a similar claim if we elide certain slots. We define a slot to be ***active*** if at least one packet is active, and we define a slot to be a ***quiet arrival*** if a packet activates but the slot is empty. The following lemma achieves our goal by ignoring slots with quiet arrivals.

Lemma 5.21. *For any integer $t > 0$, consider the first t active time slots that are not quiet arrivals. At most a $\gamma = 15/16$-fraction of these slots are empty.*

Proof. The proof is by induction on t. For the base case, observe that a quiet arrival results in an immediate reset of that packet. Thus, the first time slot in consideration is a slot during which a packet arrives and the slot is full.

For the inductive slot, we assume that the claim holds for all of the t first slots and argue that it holds at $(t+1)$th. Consider any packet u that is active at time $t+1$. Let $t_u \leq t+1$ be the slot at which u's current lifetime began. We have two cases.

Case 1. If $t_u = t+1$, then the packet just activated. By assumption, this is not a quiet arrival, and hence the $(t+1)$th slot is full. By inductive assumption, the first t slots are at most a γ-fraction empty. Concatenating these slots proves the claim.

Case 2. $t_u < t+1$. Consider the interval $[t_u, t+1]$. Since u is active for the entire interval and $t_u < t+1$, we apply Lemma 5.20 to conclude that at most a γ fraction of these slots are empty. Eliding the quiet arrivals (which are empty slots) only reinforces this claim. By inductive assumption, at most a γ fraction of the slots up to $t_u - 1$ are also empty. Concatenating these two intervals proves the claim. □

Observe that Lemma 5.21 counts all of the active interstitial slots, as any quiet arrivals are by definition part of an epoch. We thus have a way of charging the empty interstitial slots against full slots, incurring at most a $1/(1-\gamma)$ cost.

Our goal now is to bound the number of full interstitial slots, specifically the nondisrupted slots. The main idea is to show that for each full and nondisrupted slot, there is a constant probability of successful transmission. Thus, after $O(n)$ such slots, in expectation, all the packets have broadcast.

Lemma 5.22. *For a slot s, let $e_{\geq 2}$ denote the event where two or more packets are broadcast in s, and let $e_{=1}$ denote the event where one packet is broadcast in slot s. If s is an interstitial slot, then $Pr(e_{\geq 2}) = O(Pr(e_{=1}))$.*

Proof. The lemma follows immediately from Lemmas 5.1 and 5.2, since in interstitial slots, the contention is $O(1)$. □

Lemma 5.23. *There are at most $O(n)$ full, nondisrupted interstitial slots in expectation.*

Proof. By the time there are n full slots that have successful transmissions, the execution is over. And if we condition upon a given slot being full, there is a constant probability of a successful transmission by Lemma 5.22. Thus, it takes $O(n)$ such slots, in expectation, before all n packets have successfully transmitted. □

We can now prove our claims in Theorem 2.3 and Corollary 2.4 regarding expected waste and throughput.

Lemma 5.24. *If the adversary injects n packets,* Re-Backoff *has at most an expected constant factor waste.*

Proof. For the following argument, recall that λ denotes the throughput, and T is the latest any packet completes (from Section 2.2).

We will argue that the expected number of slots for all the packets to finish is $\mathrm{E}[T] = O(n + D)$ slots. We then observe that the expected nonwaste is $\mathrm{E}[\lambda] = \mathrm{E}[(n+D)/T]$. Recall Jensen's inequality: $\varphi(E[Y]) \leq E[\varphi(Y)]$ for any convex function φ and any random variable Y. Since T is positive, $1/T$ is convex, so $E[1/T] \geq 1/E[T]$. Therefore, since $E[T] \leq c'(n+D)$ for some constant $c' > 0$, we have that $E[\lambda] = (n+D)E[1/T] = E[(n+D)/T]$ is at least a constant.

Throughout the proof we consider only active slots. Reincorporating the inactive slots only increases the waste by a constant factor as a packet activates after seeing an inactive slot.

Let n_e denote the total number of slots over all nondisrupted epochs,[4] let n_d denote the number of slots over all disrupted epochs, and let n_i denote the number of full nondisrupted interstitial slots. Again, let D denote the total number of disrupted slots.

[4]Recall that an epoch is nondisrupted if $< 1/4$ of its slots are disrupted.

Our goal is to bound the number of empty interstitial slots. Lemma 5.21 implies that, ignoring some empty epoch slots (namely, the quiet arrivals), at most a δ-fraction of the remaining slots are empty. In particular, the worst case occurs if we pessimistically assume all epoch slots are full, giving at most $O(n_e + n_d + n_i + D)$ empty interstitial slots.

By Lemma 5.15, the finite bad borrower game implies that the number of epoch slots in nondisrupted epochs required to complete all n packets is $O(n)$ in expectation; therefore, $E[n_e] = O(n)$. As for the interstitial slots, Lemma 5.23 shows that $E[n_i] = O(n)$. Therefore, among nondisrupted epochs and nondisrupted interstitial slots, we conclude that the expected number of slots required for all n packets to succeed is $O(n)$.

Finally, we count the number of disrupted slots. Since a disrupted epoch is one in which at least 1/4 of the slots are disrupted, there are clearly at most $O(D)$ disrupted epoch slots. Similarly, there are at most $O(D)$ disrupted, full interstitial slots. There are also at most $O(D)$ slots in which the control channel is disrupted (which can cause wasted time on the data channel if there are no active packets). Thus, there are at most $O(D)$ such slots otherwise unaccounted for.

Thus, we conclude that there are, in expectation, $O(n)$ nondisrupted epochs and nondisrupted interstitial slots, at most $O(D)$ disrupted slots and disrupted epochs, and $O(n + D)$ empty slots. □

5.10 Interstitial Slots for Infinite Instances

We now show that the contribution from the interstitial slots does not hurt the throughput in infinite executions. To do so, we deterministically bound the contribution from the empty interstitial slots. We show that as long as we pick "measurement points" that are sufficiently large, from then on the full interstitial slots do not hurt. We use the following well-known facts about random walks (Mitzenmacher and Upfal 2005).

Fact 5.25. *Suppose that we have a biased random walk on a line with fixed step size, where the probability of going right is at least p, the probability of going left is at most $1 - p$, and the step size right is δ_r and the step size left is δ_ℓ. Suppose that $p\delta_r > (1 - p)\delta_\ell$. Then, if the random walk starts at the origin, the probability of returning to the origin is some constant strictly less than 1.*

Corollary 5.26. *For any such biased random walk, there is a last time that the walk returns to the origin.*

We use Corollary 5.26 to bound the ratio of collisions to broadcasts in nondisrupted, full interstitial rounds. From some point on, the number of collisions is always at most a constant factor of the number of broadcasts, and hence yields constant throughput. We again observe that the empty slots cannot hurt the throughput by more than a constant factor overall, because, by Lemma 5.21, at most a γ fraction of the slots in any prefix can be empty interstitial slots. Finally, we bound the disrupted interstitial slots by D. From this, we conclude that we achieve constant nonwaste and throughput, as claimed in Theorem 2.5 and Corollary 2.6:

Lemma 5.27. *In an infinite execution,* Re-Backoff *achieves at most a constant fraction of waste.*

Proof. For some slot t, let i_t^b be the number of successful broadcasts in nondisrupted interstitial slots prior to time t, and let i_t^c be the number of collisions in nondisrupted interstitial slots prior to time t.

We first argue that, with probability 1, from some point t onward, for all $t' > t$, $i_t^c \leq O(i_t^b)$. Conditioned on the fact that there is at least one broadcast in a nondisrupted interstitial round, let p be the probability of a successful broadcast and $q = 1 - p$ be the probability of a collision. We know from Lemma 5.22 that $q = O(p)$.

Define the following random walk: with probability q, take a step to the left of size 1, and with probability p, take a step to the right of size $2q/p$. Since $p \cdot (2q/p) > q$, by Corollary 5.26, we know that from some point on, this random walk is always positive.

Let t be a time slot that is after the last point where the random walk crosses the origin. We can then conclude that $i_t^c < i_t^b \cdot (2q/p)$. Since $2q/p = O(1)$, we conclude that $i_t^c = O(i_t^b)$.

Finally, we analyze the throughput. Fix any time t. Let $\hat{t}$ be the smallest time after t where the random walk defined above is positive. According to Lemma 5.16, there is a time $t' > \hat{t}$ where we have achieved constant throughput during the nondisrupted epochs; i.e., a constant fraction of the slots in nondisrupted epoch are broadcasts. By the analysis of the random walk, we conclude that a constant fraction of the full, nondisrupted interstitial rounds are broadcasts. By assumption, at most $O(D)$ slots are part of disrupted epochs, and there are at most D disrupted interstitial slots. There are at most D disrupted control slots (which may cause delays on the data channel if there are no active packets). Finally, by Lemma 5.21, at most γ of the data channel slots in the entire execution are empty interstitial slots. Putting these pieces together yields constant throughput overall. □

6 ANALYSIS OF THE NUMBER OF ACCESS ATTEMPTS

In this section, we analyze the number of access attempts. We show that in the absence of disruption, the number of broadcasts is small and the adversary requires a significant amount of disruption to cause even a small increase in the number of access attempts.

We first analyze how often a packet resets, showing that it is likely to succeed before it has a chance to reset. This ensures that a packet cannot be forced to make a large number of attempts via repeated resets.

Lemma 6.1. *When a packet becomes active, it succeeds (instead of resetting) with constant probability at least* $1 - e^{-d/4}$.

Proof. In this proof, we grant the adversary even more power than given by the model—in each slot, the adversary is allowed to specify whether the slot is "covered," meaning that it is either disrupted or some other packet transmits. The only thing the adversary does not control is the packet in question.

Starting from the time the packet becomes active, we divide time into windows $W_0, W_1, W_2, W_3, \ldots$, where window W_i has length 2^i. Note that if the first slot is covered, the packet cannot possibly reset until time 16 or later, which more than subsumes window W_1. Similarly, if at least one slot is also covered in window W_1, then the packet cannot possibly reset until after window W_2. In general, if at least half the slots are covered in each of the windows $W_0, W_1, \ldots, W_{i-1}$, then the packet either stays alive through W_i or succeeds sometime before the end of W_i—it cannot reset.

Our argument thus proceeds inductively over windows, stopping at the first window W_i that is not at least half covered. Window W_i has 2^i slots. In the first slot, the probability of a packet u sending is d/s_u where $s_u \geq 2^i - 1$, while in the final slot of this window, $s_u \geq 2^{i+1} - 1$, so the probability of u sending is at least $d/2^{i+1}$. Thus, in window W_i, the packet transmits independently in each data slot with probability at least $d/2^{i+1}$, where d is a constant specified in the protocol. Thus, if at least 2^{i-1} of the slots in W_i are left uncovered, the packet has either succeeded earlier or succeeds in window W_i with probability at least $1 - (1 - d/2^{i+1})^{2^{i-1}} \geq 1 - 1/e^{d/4}$, which is constant. □

Corollary 6.2. *For any positive integer k, the probability that a packet resets k times is at most* $1/e^{\Theta(k)}$.

Proof. The only observation we need is that for a particular packet, each of its lifetimes are nonoverlapping. Thus, Lemma 6.1 bounds the probability of a reset for a given lifetime, and for each lifetime the probabilities are independent. Thus, each reset occurs with probability at most $1/e^{d/4}$, and hence the probability of k resets is at most $1/e^{kd/4} = 1/e^{\Theta(k)}$. □

We can now bound the total number of access attempts that a packet makes during the first t slots after its arrival. If it has not yet reset by time t, it is easy to see that it has made $O(\log^2 t + 1)$ access attempts in expectation—and we have shown above that a packet is unlikely to reset too many times. This yields:

Lemma 6.3. *In the first t slots following a packet's arrival, it makes $O(\log^2(t) + 1)$ attempts in expectation and $O(\log^2(t) \cdot \log n)$ attempts with high probability.*

Proof. Consider any lifetime of the packet. The expected number of access attempts during a slot is equal to the packet's transmission probability, and hence the expected total number of access attempts is the sum of probabilities across all slots by linearity of expectation. The expected number of access attempts in a lifetime is thus at most $\sum_{s=1}^{t} \Theta(1 + \ln s)/s = \Theta(\log^2 t + 1)$, with the $\Theta(1 + \ln s)$ arising from the higher transmission probability of $c \max(\ln s, 1)/s$ in control slots.

We now compute the expected number of access attempts made by the packet by using linearity of expectation across all lifetimes of the packet. In particular, the number of access attempts during the k^{th} lifetime is 0 if the packet does not reset $k - 1$ times, and hence applying Corollary 6.2, the expected number of access attempts of the k^{th} lifetime is $1/e^{\Theta(k)} \cdot O(\log^2 t + 1)$. Using linearity of expectation across all lifetimes, we get a total expected number of access attempts of $O(\log^2 t + 1)$ $\sum_{k=0}^{\infty} 1/e^{\Theta(k)} = O(\log^2 t + 1)$.

Finally, by letting $k = c' \ln n$ for some sufficiently large constant $c' > 0$, we have by Corollary 6.2 that the total number of access attempts is $O(\log^2(t) \cdot \log n)$ with high probability. □

We can now prove our claims in Theorems 2.3 and 2.5 regarding the expected number of access attempts per packet. In the finite case, we have already bounded the expected length of the execution in Lemma 5.24. In the infinite case, we separately analyze the disrupted slots, the nondisrupted slots with young packets, and the nondisrupted slots with old packets, bounding the number of attempts.

Lemma 6.4. *Consider the finite case, let n be the number of injected packets, and let D be the number of disrupted slots. Then the expected number of access attempts each packet makes is $O(\log^2(n + D) + 1)$.*

Proof. Suppose the execution completes in time t. Then applying Lemma 6.3 yields an expected number of access attempts of $O(\log^2 t + 1)$, as the packet must complete before the execution completes. Let T be the expected time of completion. From Markov's inequality, $t \geq T^i$ with probability at most $1/T^{i-1}$. Summing across all i, the expected number of access attempts becomes $O(\sum_{i=1}^{\infty}(1/T^i)(\log^2(T^i) + 1)) = O(\log^2 T + 1) \cdot \sum_{i=1}^{\infty}(i^2/T^i) = O(\log^2 T)$. Substituting in the expected makespan $T = O(n + D)$ from Lemma 5.24 concludes the theorem. □

Lemma 6.5. *Consider any time t in the infinite case at which we have λ throughput, for constant λ. Let D_t be the total number of disrupted slots before t, and let η_t be the maximum contention prior to time t. Then the expected average number of access attempts per packet active prior to time t is $O(\log^2(\eta_t + D_t) + 1)$.*

Proof. The analysis is split into two cases: either $D_t \geq \lambda t/2$ or $D_t < \lambda t/2$. The first case is easy—Lemma 6.3 states that the expected number of access attempts per packet is $O(\log^2 t + 1) = O(\log^2 D_t + 1)$.

Suppose for the remainder that $D < \lambda t/2$. Then we divide the analysis here into three parts: (A) the disrupted slots, (B) the nondisrupted slots for "young" packets, and (C) the nondisrupted slots for "old" packets. In parts A and B, we shall show that the expected number of access attempts per packet is at most $O(\log^2 D_t + \log^2 \eta_t) = O(\log^2(D_t + \eta_t))$, regardless of whether we have constant throughput. It is only in part C that we leverage the assumption that time t is a time at which we have constant throughput.

A. Consider a single lifetime of a specific packet. Our goal is to bound the number of access attempts by this packet during all D_t disrupted slots. The number of access attempts is maximized if all D_t slots occur as early as possible in the packet's lifetime, in which case the expected number of access attempts during disrupted slots (i.e., during the first D_t slots of its lifetime) is at most $O(\log^2 D_t + 1)$ per lifetime (Lemma 6.3). As in Lemma 6.3, we then apply Corollary 6.2 and linearity of expectation to get an expected number of access attempts of at most $O(\log^2 D_t + 1) \sum_{k=0}^{\infty} 1/e^{\Theta(k)} = O(\log^2 D_t + 1)$.

B. Consider a specific packet. We say that the packet is young during the first η_t^2 slots of its lifetime, during which it makes $O(\log^2 \eta_t^2 + 1) = O(\log^2 \eta_t + 1)$ access attempts in expectation (Lemma 6.3). Summing across all lifetimes as above, we conclude that the contribution for young packets is $O(\log^2 \eta_t + 1)$.

C. If a packet is not young, i.e., if its age is at least η_t^2, we say that it is old. Unlike parts A and B, which analyze on a per-packet basis, this part analyzes the number of access attempts in aggregate. For every nondisrupted slot, there are at most η_t packets in the system, and hence at most η_t packets are old. Each old packet transmits with probability at most $O((1 + \ln(\eta_t^2))/\eta_t^2) = O(1/\eta_t)$, and hence the expected number of access attempts across all old packets in any slot is $O(1)$. Using linearity of expectation across all t slots, we have a total expected number of access attempts by old packets of $O(t)$.

Since we have constant throughput at time t, we know that at least λt slots are either disrupted or successful transmissions. There are at most $\lambda t/2$ disrupted slots by assumption, and hence there are at least $\lambda t/2 = \Omega(t)$ successful transmissions. We charge the $O(t)$ number of access attempts from old packets to these $\Omega(t)$ completed packets, for an additional $O(1)$ number of access attempts per successful packet. □

7 SYNCHRONIZATION: REDUCING TO ONE CHANNEL

In this section, we describe how to transform the Re-Backoff algorithm so that it runs on a single channel. We first describe the modified algorithm. We then show that it maintains a synchronized view of the channel. Finally, we review the analysis, highlighting those few places where it has to be modified for this variant.

7.1 Modified Algorithm

We begin by describing the modified algorithm that uses only one channel.

As in Re-Backoff, packets are initially inactive. They monitor the channel and wait to hear *two* empty slots, immediately after which they become active. This is in contrast to the original specification that a packet becomes active in the next slot when it hears one empty *control* slot.

The main idea is that once a packet becomes active, it alternates executing *control slots* and *data slots*. This idea is refined below to synchronize packets so that they agree on the parity of the slots.

When a packet becomes active, it treats its first slot as a control slot. In that first control slot, it broadcasts *with probability 1*. It then proceeds to alternate data and control slots. From then on, as in the original specification, a packet u with age s_u broadcasts in a control slot with probability $c \max\{\ln s_u, 1\}/s_u$, and in a data slot with probability d/s_u, terminating upon success, and where c and d are constants as before.

For example, if packet u is inactive and it observes slot s and $s + 1$ to be empty, it becomes active in slot $s + 2$. It treats $s + 2$ as a control slot and sends a control signal with probability 1. It then alternates, treating $s + 3$ as a data slot, $s + 4$ as a control slot, and so forth.

A packet calculates its age as follows: prior to its first active slot (which, again, it designates a control slot), the packet sets its age to 1; immediately before every subsequent control slot, it increments its age. That is, the age of a packet in slot s is the number of slots that it has designated as control slots since it became active, up to and including slot s.

With no further synchronization, different packets may treat a given slot as a control slot and a data slot at the same time. We thus add a synchronization mechanism:

If a packet observes an empty control slot followed by a full data slot, then it designates the following slot as a data slot.

In doing so, it breaks the rule of alternation, synchronizing the packets.

For example, suppose packet u believes that s is a control slot. If s is empty (i.e., no broadcast, no collision, no disruption), and if $s + 1$ is full (i.e., a broadcast or a collision or disruption), then packet u treats slot $s + 2$ as a data slot. It then continues to alternate, treating $s + 3$ as a control slot.

If a packet completes in a data slot that immediately follows an empty control slot, then it does not terminate immediately. Instead, the packet participates in the additional data slot that follows—by sending its data with the same probability as in the first data slot—before the packet terminates.

Finally, recall that a packet resets if it finds that a γ-fraction of data slots have been empty since it became active. Here, the reset rule remains identical with one change: if there are two consecutive data slots, then the packet does not count the results from the first data slot.

For example, if s is a control slot and $s + 1$ and $s + 2$ are data slots (because s is empty and $s + 1$ is full), then after these three slots: if $s + 2$ is empty, a packet increments its count of empty data slots by 1; if $s + 2$ is full, a packet increments its count of full data slots by 1.

7.2 Slot Agreement

We observe that packets agree on whether a slot is a control or a data slot; i.e., synchronization works:

Lemma 7.1. *Let t be a slot, and let u and v be two packets that are active in slot $t - 1$ and slot t. Then u considers t a control slot if and only if v considers t a control slot. Similarly, u considers t a data slot if and only if v considers t a data slot.*

Proof. We consider three exhaustive cases:

Case 1: Assume u and v are injected in the same slot $t_0 < t$. Then u and v both consider t_0 to be a control slot and observe the same pattern of full and empty slots from then on. Hence, u and v continue to identify slots in the same manner.

Now assume instead that u is injected in slot t_u and v is injected in slot $t_v > t_u$ (where $t_v < t$). There are two cases to consider:

Case 2. Assume that u considers slot t_v to be a control slot. In that case, as of slot t_v, both u and v consider t_v to be a control slot and both observe the same pattern of full and empty slots from then on. Hence, u and v continue to identify slots in the same manner.

Case 3. Assume that u considers slot t_v to be a data slot. We make two observations:

— *Observation 1.* Slots $t_v - 1$ and $t_v - 2$ are empty, since packet v only becomes active after observing two empty slots.

— *Observation 2.* Packet v broadcasts in slot t_v, since a packet broadcasts in its first control slot with probability 1, and so t_v is full.

Using these observations, we make a critical claim: u must view $t_v - 1$ as a control slot.

Why? Assume otherwise, that u views slot $t_v - 1$ as a data slot, and we will derive a contradiction. By assumption in Case 3, packet u views slot t_v as a data slot; therefore, u witnesses two consecutive data slots $t_v - 1$ and t_v. This implies that $t_v - 1$ must be full; otherwise, if slot $t_v - 1$ is empty, there cannot be two consecutive data slots ($t_v - 1$ and t_v), which would yield a contradiction. But assuming that slot $t_v - 1$ is full contradicts the requirements for packet v being activated, as specified in Observation 1.

By the above argument, packet u views $t_v - 1$ as a control slot and, furthermore, this control slot is empty (again, otherwise v cannot be active). Thus, packet u views t_v as a data slot, and by Observation 2, this data slot is full. This implies that packet u views slot $t_v + 1$ as a data slot too. Packet v also considers $t_v + 1$ to be a data slot, since it is alternating slot types. Hence, as of slot $t_v + 1$, both u and v agree on the slot designation. From that point on, but packet u and v observe the same pattern of full and empty slots, and hence they continue to identify slots in the same manner. □

As a result of Lemma 7.1, we can officially designate slots as control or data slots. Consider a slot t:

— If no packet is active in slot t, we designate slot t to be an empty slot.
— If there exists any packet u that is active in slot $t - 1$ and slot t, then we designate slot t a data or control slot based on the designation of packet u.
— If all the packets active in slot t were *not* active in slot $t - 1$, then we designate slot t a control slot (as do all the newly activated packets).

We can then designate a pair or triple of slots consisting of a control slot followed by one or two data slots as a ***slot-group***. Throughout, unless specified otherwise, when we refer to a slot as a control or a data slot, we refer to this global (synchronized) designation.

7.3 Changes to the Analysis

We can then repeat the analysis found in Section 5 and 6, with a very small number of minor changes, yielding the same waste, throughput, and energy results. Here, we highlight these issues.

The analysis begins by defining contention, dividing packets into young and old packets, defining control failures, and so forth. Nothing needs to change as a result. It is worth noting that a control failure is no longer a sufficient condition for a packet to become active (as two consecutive slots are needed), but it remains a necessary condition.

We proceed to calculate the probability of various events as a function of contention (e.g., Lemma 5.1 and Lemma 5.2), which remain identical for all slots in which no packet becomes active.

We define epochs and streaks as before; i.e., an epoch begins at the beginning of a slot group in which some packet is activated, and a streak consists of the subsequent σ_t slot-groups. If a streak ends with contention below 32 (notice that we have increased the contention limit for an epoch), then the epoch ends and interstitial slots begin; otherwise, a new streak begins.

We say an epoch is disrupted if at least 1/4 of its *data slots* are disrupted. For this purpose, if a slot-group contains three slots, we count only the second data slot.

The analysis of contention remains unchanged (e.g., Lemma 5.4 and Lemma 5.5), with the only difference that, due to the extra data slots, more packets can complete, lowering the contention twice as fast. Thus, we assume that initially $X(t) \geq 16$, and conclude that it does not drop by more than a factor of 16. Similarly, the analysis of control failures (Lemma 5.7 and Lemma 5.8) remains unchanged.

The analysis of resets, however, requires a small modification. Consider Lemma 5.10, which argues that a reset occurs during an epoch with at most constant probability $1 - q'$ for some constant $q' > 0$ depending only on d.

Here, we need to modify the proof slightly. Consider a slot-group with an empty control slot. There are two cases that lead to us "counting" an empty data slot (and increasing the likelihood of reset): (1) the data slot is empty, and (2) the data slot is full, followed immediately by an empty data slot. The first full data slot triggers another data slot (for synchronization purposes), which offers a second chance for an empty data slot. Recall that we count a data slot as empty for a slot-group in a three-slot slot-group if the second data slot is empty.

Thus, we modify the proof as follows. First, we consider epochs whose initial contention is 32. This ensures that through the epoch, the contention is at least 2, and hence the probability of an empty data slot is at most e^{-2d} (by Lemma 5.2).

As such, the probability that, for a given slot-group, we count a data slot as empty is at most $2e^{-2d}$. Hence, for the purpose of arguing the analog to Lemma 5.10, we define $p = 1 - 2e^{-d}$ as the probability of success. The remainder of the analysis continues as before, with the conclusion that at least $p/4 \geq 1 - \gamma$ fraction of the slots in the epoch are full with probability at least q'.

The remainder of the throughput analysis continues unchanged. For example, Lemma 5.12 is a critical lemma that shows that every streak is successful with constant probability. This follows from a calculation based on the probability of control failures and resets. As those bounds remain unchanged, the lemma holds unchanged here. As a result, Corollary 5.13 also holds unchanged, where lengths refer to the number of slot-groups (instead of the number of slots).

Similarly, the analysis of the bad borrower game (Lemma 5.14, Lemma 5.15, Lemma 5.16, Corollary 5.19) is not impacted by the modifications.

It remains to consider the interstitial slots. Lemma 5.20 shows that for any prefix of a packet's lifetime (as long as it is length at least 2), at least a $(1 - \gamma)$ fraction of the data slots are full. This remains true, so long as we count only the second data slot in a three-slot slot-group.

Lemma 5.21 then follows, implying that for time slots that are nonquiet arrivals, at most a γ-fraction of the data slots are empty. Again, this holds identically, so long as we only count the second data slot in a three-slot slot-group. (Moreover, the first data slot in a three-slot slot-group is always full, by definition.)

At this point, we have accounted for all the empty interstitial slots and the remaining accounting continues as before. We conclude Lemma 5.23 exactly as before; i.e., there are $O(n)$ full nondisrupted interstitial slots. Putting the pieces together, this yields Theorem 2.3, which shows expected constant throughput in finite executions.

The analysis of interstitial slots in the infinite case proceeds similarly, with Lemma 5.27 following as before from the same basic analysis of random walks (which does not depend on the protocol itself and hence is unchanged).

Finally, the energy analysis is unchanged, beyond accounting for the energy used in the extra data slots. A packet may now spend somewhat more energy, as it may now broadcast in two data slots for some slot-groups instead of just one. This change, however, increases the energy usage by at most a constant factor.

8 CONCLUSION AND FUTURE WORK

We have presented a simple backoff protocol that provides constant expected throughput in the face of online packet arrivals. Our protocol is robust to worst-case interference on the communication channel.

One issue not explored is "cohabiting" (or "coexisting") networks, where different applications are sharing the same communication channel. If these applications are unaware of each other (e.g.,

treating messages from the other application as noise, or as sent by an adversary), then existing backoff protocols and MAC layers (e.g., Awerbuch et al. (2008)) no longer guarantee good throughput. By contrast, since the algorithm here has no sensitivity to which messages are received—only monitoring for when the channel is occupied or free—it will still guarantee constant throughput, collectively, in such settings. (See Richa et al. (2012), who address the problem of coexisting networks.)

Currently, our results rely on all participants agreeing on the start time of each slot. However, it may be interesting to consider the situation where any two participants may differ by at most a quantity $\delta > 0$ in their belief of when a slot starts. If δ is a known, bounded quantity, then each participant may simply widen its original slot size by δ at the start and at the end; this enlarged slot time provides a sufficient buffer to accommodate potential disagreement. If a participant has a packet to send, the participant sends as it normally would using its original notion of when the slot starts. Otherwise, if a participant wishes to listen, this is done throughout the enlarged slot. However, if $\delta > 0$ is unknown or unbounded, then the problem appears challenging and we speculate new techniques would be required to periodically synchronize the participants.

In practice, for Internet-connected devices where frequent requests for clock updates are feasible, synchronization is possible to within milliseconds and even microseconds (see the Network Time Protocol (Mills 1991)). In wireless ad hoc networks, the situation may be more challenging due to constraints on power and bandwidth; however, synchronization may still be feasible (Li et al. 2005; Ganeriwal et al. 2003). Under a jamming adversarial, synchronization guarantees are provided in Ganeriwal et al. (2008); however, these results rely on message authentication and the ability to securely establish system membership.

Another open problem is whether we can tolerate less predictable noise, i.e., a more powerful adversary that can cause different packets to observe different conditions (i.e., an n-uniform adversary). Such a model better captures a multihop network where noise may come from different directions. The key challenge is that packets may see different slots as empty, and hence reset at different times. While resets are further desynchronized, it remains the case that there cannot be too many truly empty slots (i.e., empty for everyone) without causing resets. Hence, again, we suspect that the results of this article hold, with small changes.

Finally, we are interested in whether our results extend to multihop networks. Previous work on jamming resistance (Richa et al. 2013b) has shown that it is possible to design a jamming-resistant backoff protocol that continues to guarantee good throughput, even when the devices sending the packets are distributed over a large area.

REFERENCES

David J. Aldous. 1987. Ultimate instability of exponential back-off protocol for acknowledgment-based transmission control of random access communication channels. *IEEE Transactions on Information Theory* 33, 2 (1987), 219–223.

Lakshmi Anantharamu, Bogdan S. Chlebus, Dariusz R. Kowalski, and Mariusz A. Rokicki. 2011. Medium access control for adversarial channels with jamming. In *Proceedings of the 18th International Colloquium on Structural Information and Communication Complexity (SIROCCO'11)*. 89–100.

Lakshmi Anantharamu, Bogdan S. Chlebus, and Mariusz A. Rokicki. 2009. Adversarial multiple access channel with individual injection rates. In *Proceedings of the 13th International Conference on Principles of Distributed Systems (OPODIS'09)*. 174–188.

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

Baruch Awerbuch, Andrea Richa, and Christian Scheideler. 2008. A jamming-resistant MAC protocol for single-hop wireless networks. In *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC'08)*. 45–54.

Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. 1999. Balanced allocations. *SIAM Journal on Computing* 29, 1 (Sept. 1999), 180–200.

Michael A. Bender, Martin Farach-Colton, Simai He, Bradley C. Kuszmaul, and Charles E. Leiserson. 2005. Adversarial contention resolution for simple channels. In *Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'05)*. 325–332.

Michael A. Bender, Jeremy T. Fineman, and Seth Gilbert. 2006. Contention resolution with heterogeneous job sizes. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA'06)*. 112–123.

Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Maxwell Young. 2016a. How to scale exponential backoff: Constant throughput, polylog access attempts, and robustness. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'16)*. 636–654.

Michael A. Bender, Tsvi Kopelowitz, Seth Pettie, and Maxwell Young. 2016b. Contention resolution with log-logstar channel accesses. In *Proceedings of the 48th Annual Symposium on the Theory of Computing (STOC'16)*. 499–508.

Petra Berenbrink, Artur Czumaj, Matthias Englert, Tom Friedetzky, and Lars Nagel. 2012. Multiple-choice balanced allocation in (almost) parallel. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX-RANDOM'12)*. 411–422.

Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. 2006. Balanced allocations: The heavily loaded case. *SIAM Journal on Computing* 35, 6 (2006), 1350–1385.

Petra Berenbrink, Kamyar Khodamoradi, Thomas Sauerwald, and Alexandre Stauffer. 2013. Balls-into-bins with nearly optimal load distribution. In *Proceedings of the 25th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'13)*. 326–335.

Daniel J. Bernstein. 1998. qmail — An email Message Transfer Agent. Retrieved from http://cr.yp.to/qmail.html.

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

Giuseppe Bianchi and Ilenia Tinnirello. 2003. Kalman filter estimation of the number of competing terminals in an IEEE 802.11 network. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'03)*, Vol. 2. 844–852.

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

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

John I. Capetanakis. 1979. Generalized TDMA: The multi-accessing tree protocol. *IEEE Transactions on Communications* 27, 10 (Oct. 1979), 1476–1484.

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

Bogdan S. Chlebus, Leszek Gasieniec, Dariusz R. Kowalski, and Tomasz Radzik. 2005. On the wake-up problem in radio networks. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*. 347–359.

Bogdan S. Chlebus and Dariusz R. Kowalski. 2004. A better wake-up in radio networks. In *Proceedings of 23rd ACM Symposium on Principles of Distributed Computing (PODC'04)*. 266–274.

Bogdan S. Chlebus, Dariusz R. Kowalski, and Mariusz A. Rokicki. 2006. Adversarial queuing on the multiple-access channel. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC'06)*. 92–101.

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

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

Richard Cole, Alan M. Frieze, Bruce M. Maggs, Michael Mitzenmacher, Andréa W. Richa, Ramesh K. Sitaraman, and Eli Upfal. 1998. On balls and bins with deletions. In *Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM'98)*. 145–158.

Bryan Costales and Eric Allman. 2002. *Sendmail* (3rd ed.). O'Reilly.

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

Jeremy T. Fineman, Calvin Newport and Tonghe Wang. Contention resolution on multiple channels with collision detection. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'16)*. 175–184.

Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. 2003. Timing-sync protocol for sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*. 138–149.

Saurabh Ganeriwal, Christina Pöpper, Srdjan Čapkun, and Mani B. Srivastava. 2008. Secure time synchronization in sensor networks. *ACM Transactions on Information and System Security* 11, 23 (2008), 1–35.

Mihály Geréb-Graus and Thanasis Tsantilas. 1992. Efficient optical communication in parallel computers. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'92)*. 41–48.

Leslie Ann Goldberg, Mark Jerrum, Tom Leighton, and Satish Rao. 1993. A doubly logarithmic communication algorithm for the completely connected optical communication parallel computer. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'93)*. 300–309.

Leslie Ann Goldberg and Philip D. MacKenzie. 1996. Analysis of practical backoff protocols for contention resolution with multiple servers. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'96)*. 554–563.

Leslie Ann Goldberg, Philip D. MacKenzie, Mike Paterson, and Aravind Srinivasan. 2000. Contention resolution with constant expected delay. 47, 6 (Nov. 2000), 1048–1096.

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

Google. 2014. GCM (Google Cloud Messaging) Advanced Topics. Retrieved from http://developer.android.com/google/gcm/adv.html#retry.

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

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

Zygmunt J. Haas and Jing Deng. 2002. Dual busy tone multiple access (DBTMA) - A multiple access control scheme for ad hoc networks. *IEEE Transactions on Communications* 50, 6 (June 2002), 975–985.

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

Maurice Herlihy and J. Eliot B. Moss. 1993. Transactional memory: Architectural support for lock-free data structures. In *Proceedings of the 20th International Conference on Computer Architecture*. 289–300. Retrieved from http://www.cs.brown.edu/people/mph/isca2.ps.

Alefiya Hussain, John Heidemann, and Christos Papadopoulos. 2003. A framework for classifying denial of service attacks. In *2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'03)*. 99–110.

Van Jacobson. 1988. Congestion avoidance and control. *SIGCOMM Computer Communication Review* 18, 4 (Aug. 1988), 314–329.

Tomasz Jurdzinski and Grzegorz Stachowiak. 2015. The cost of synchronizing multiple-access channels. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'15)*. 421–430.

Marek Klonowski and Dominik Pajak. 2015. Electing a leader in wireless networks quickly despite jamming. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures*. 304–312.

James F. Kurose and Keith Ross. 2002. *Computer Networking: A Top-Down Approach Featuring the Internet* (2nd ed.). Addison-Wesley Longman Publishing Co., Boston, MA.

Byung-Jae Kwak, Nah-Oak Song, and Leonard E. Miller. 2005. Performance analysis of exponential backoff. *IEEE/ACM Transactions on Networking* 13, 2 (2005), 343–355.

Yuan Li, Wei Ye, and John Heidemann. 2005. Energy and latency control in low duty cycle MAC protocols. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'05)*. 676–682.

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

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

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

David L. Mills. 1991. Internet time synchronization: The network time protocol. *IEEE Transactions on Communications* 39, 10 (1991), 1482–1493.

Michael Mitzenmacher. 2001. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems* 12, 10 (Oct. 2001), 1094–1104.

Michael Mitzenmacher and Eli Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

Michael Mitzenmacher, Andréa W. Richa, and Ramesh Sitaraman. 2001. The power of two random choices: A survey of techniques and results. *Combinatorial Optimization* 9 (2001), 255–304.

Amit Mondal and Aleksandar Kuzmanovic. 2008. Removing exponential backoff from TCP. *SIGCOMM Computer Communication Review* 38, 5 (Sept. 2008), 17–28.

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

Google Apps Platform. 2011. Google Documents List API version 3.0: Implementing Exponential Backoff. Retrieved from https://developers.google.com/google-apps/documents-list/?csw=1#implementing_exponential_backoff.

Prabhakar Raghavan and Eli Upfal. 1995. Stochastic contention resolution with short delays. In *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing (STOC'95)*. 229–237.

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

Ravi Rajwar and James R. Goodman. 2001. Speculative lock elision: Enabling highly concurrent multithreaded execution. In *Proceedings of the 34th Annual International Symposium on Microarchitecture*. 294–305. Retrieved from http://www.cs.wisc.edu/ rajwar/papers/micro01.pdf.

Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. 2010. A jamming-resistant MAC protocol for multi-hop wireless networks. In *Proceedings of the International Symposium on Distributed Computing (DISC'10)*. 179–193.

Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. 2011. Competitive and fair medium access despite reactive jamming. In *Proceedings of the 31st International Conference on Distributed Computing Systems (ICDCS'11)*. 507–516.

Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. 2012. Competitive and fair throughput for co-existing networks under adversarial interference. In *Proceedings of the 31st ACM Symposium on Principles of Distributed Computing (PODC'12)*. 291–300.

Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. 2013b. Competitive throughput in multi-hop wireless networks despite adaptive jamming. *Distributed Computing* 26, 3 (2013), 159–171.

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

Amazon Web Services. 2012. Error Retries and Exponential Backoff in AWS. Retrieved from http://docs.aws.amazon.com/general/latest/gr/api-retries.html.

Cheng Shong Wu and Victor O.K. Li. 1988. Receiver-initiated busy-tone multiple access in packet radio networks. In *Proceedings of the ACM Workshop on Frontiers in Computer Communications Technology (SIGCOMM'87)*. 336–342.

Nah-Oak Song, Byung-Jae Kwak, and Leonard E. Miller. 2003. On the stability of exponential backoff. *Journal of Research of the National Institute of Standards and Technology* 108, 4 (2003), 289–297.

Berthold Vöcking. 2003. How asymmetry helps load balancing. *Journal of the ACM* 50, 4 (2003), 568–589.

John Paul Walters, Zhengqiang Liang, Weisong Shi, and Vipin Chaudhary. 2007. Wireless sensor network security: A survey. In *Security in Distributed, Grid, Mobile, and Pervasive Computing*. Auerbach Publications.

Dan E. Willard. 1986. Log-logarithmic selection resolution protocols in a multiple access channel. *SIAM Journal on Computing* 15, 2 (May 1986), 468–477.

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

Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. 2005. The feasibility of launching and detecting jamming attacks in wireless networks. In *ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc'05)*. 46–57.

Received August 2016; revised August 2018; accepted September 2018