Adversarial Multiple Access Channels
with Individual Injection Rates [*]

Lakshmi Anantharamu [†]   Bogdan S. Chlebus [‡]   Mariusz A. Rokicki [§]

Abstract

We study deterministic distributed broadcasting in synchronous multiple-access channels. Packets are injected into n nodes by a window-type adversary that is constrained by a window w and injection rates individually assigned to all nodes. We investigate what queue size and packet latency can be achieved with the maximum aggregate injection rate of one packet per round, depending on properties of channels and algorithms. We give a non-adaptive algorithm for channels with collision detection and an adaptive algorithm for channels without collision detection that achieve $\mathcal{O}(\min(n+w, w\log n))$ packet latency. We show that packet latency has to be either $\Omega(w\max(1, \log_w n))$, when $w \leq n$, or $\Omega(w+n)$, when $w > n$, as a matching lower bound to these algorithms. We develop a non-adaptive algorithm for channels without collision detection that achieves $\mathcal{O}(n+w)$ queue size and $\mathcal{O}(nw)$ packet latency. This is in contrast with the adversarial model of global injection rates, in which non-adaptive algorithms with bounded packet latency do not exist (Chlebus et al. *Distributed Computing* 22(2): 93–116, 2009). Our algorithm avoids collisions produced by simultaneous transmissions; we show that any algorithm with this property must have $\Omega(nw)$ packet latency.

Key words: multiple access channel, dynamic broadcasting, adversarial queuing, distributed algorithm, deterministic algorithm, stability, packet latency.

1 Introduction

A multiple access channel is a model of a communication environment supporting broadcasting among a set of nodes. What defines such a network is the property that a transmission by a node is heard by every node in the system precisely when this transmission does not overlap with other transmissions. Overlapping multiple transmissions collide with one another so that none can be received successfully.

There are two popular representations of multiple-access channels. One representation is provided by single-hop radio networks. These are wireless networks in which nodes use one radio frequency and a transmission by a node can be sensed by any other node. In such networks, overlapping transmissions interfere with one another, which results in receiving the contents attempted to be communicated as garbled, while a single transmission is successfully received by every node. Another representation is provided by the implementation of local area networks by a wired Ethernet, as represented by the IEEE 802.3 collection of standards. Abstracting multiple access channels from the medium access control of such communication environments allows one to study the optimality of communication algorithms in an idealized but precisely defined algorithmic communication setting, without the constraints of continually evolving wireless technologies and IEEE standards.

We consider the synchronous slotted model of communication, in which executions are partitioned into rounds determined by a global clock. When a transmission is successful, then the transmitted message is immediately received by every other node attached to the multiple-access channel. Messages transmitted in one round by different nodes are considered as overlapping and so colliding with one another. With these stipulations, at most one message can be successfully broadcast in a round.

Performance of broadcast algorithms can be measured in terms of various natural metrics. Among them, queue size and packet delay are the most often considered. Stability of a system means that the number of packets stored in local queues is bounded in all rounds. An apparently weakest expectation about the amount of time spent by packets in the system is that every packet is eventually successfully transmitted, see [22] and [46]. Packet latency denotes the maximum time that a packet may spend in a queue. Even when every packet is eventually successfully transmitted, there may be no finite bound on the times that packets spend waiting in queues; when a finite bound exists then the system is stable.

It is natural to consider packet generation as restricted by two parameters: (1) an average frequency of injection at nodes and (2) an upper bound on the number of packets that can be injected in one round. The frequency of packet injections into the system is called injection rate. Such frequencies are typically determined by the statistical constraints when modeling and simulating networks; in this paper we do not adhere to such approaches. The number of packets that can be injected into the system in one round is often called the burstiness of traffic.

Using randomization is a natural means to implement efficient arbitration for access to the channel; popular randomized algorithms include Aloha [1] and backoff ones [14, 35, 38, 43]. We consider deterministic algorithms that do not resort to randomization. We use adversarial queuing as the methodology to study the worst-case performance of these deterministic communication algorithms. Adversarial queuing is based on defining packet injection rates as upper bounds on the average number of packets inserted into the system. The averages of the numbers of injections are

defined without resorting to stochastic notions.

Recent work on adversarial queuing for multiple access channels, see [5, 22, 23], concerned adversaries constrained by "global" injection rates, in the sense that the adversary is restricted by the number of packets injected into all nodes but not by how many packets are injected into any specific node. We consider adversaries with individual rates associated with nodes; in this model the adversary's injection rate is constrained separately for each node. An adversary with injection rates associated with individual nodes is more restricted than one with a "global" injection rate. We understand the throughput of an algorithm as the maximum (global) rate for which it is stable. One may observe that the (global) injection rate of one packet per round is the maximum rate that allows for a stable algorithm. It follows that the throughput of any algorithm is at most 1.

A definition of injection rates depends on how we average injections. There are two popular approaches to define averaging, which produce two corresponding classes of adversaries, called leaky-bucket and window ones. The former class comprises general adversaries for which all the time intervals are used to constrain injection rate by averaging over them. The latter class restricts intervals over which averages are taken to be of the same "window" length, which is a parameter of an adversary. When injection rates are smaller than 1 then the two adversarial models of "global" injection rates are equivalent, as shown by Rosén [45].

In the context of adversarial queuing in store-and-forward networks, "globally constrained" window adversaries were first used by Borodin et al. [19] and similar leaky-bucket ones by Andrews et al. [9]. For the multiple access channel, Chlebus et al. [23] used window adversaries with injection rates smaller than 1, which does not restrict the generality of results for such rates. Chlebus et al. [22] used both models for "global" injection rate 1; for such highest possible "global" rate, the window adversarial model is strictly weaker, see [22, 45].

We consider constraints on adversaries formulated as restrictions on what can be injected separately at each node and also by what can be injected into all the nodes combined. Constraints on adversaries are *global* if they are formulated in terms of the numbers of packets injected in suitable time intervals, without any concern about the nodes in which the packets are injected. Traffic constraints are *local* when the patterns of injection are considered separately for each node. Constraints for the whole network implied by local ones are called *aggregate* in this paper. In particular, we may have local and aggregate and global burstiness, and local and aggregate and global injection rates. Observe that global constraints are logically weaker than aggregate constraints. Global and aggregate injection rate 1 is the maximum that a channel can handle in a stable way, since stability provides that the throughput rate is at least as large as the injection rate. Adversaries with global injection rates were used in [22, 23] to model traffic requirements in multiple access channels. In this paper, we introduce adversaries with local injection rates.

To categorize algorithms, we use the terminology similar to that in in [22, 23]. Algorithms may either use control bits piggybacked on transmitted packets or not; when they do, then they are called adaptive.

A summary of the results. We study effectiveness of broadcasting when traffic demands are specified as adversarial environments with individual injection rates associated with nodes and when algorithms are both distributed and deterministic. We allow the adversaries to be such that the associated aggregate injection rate (the sum of all the individual rates) is 1, which is the maximum value allowing for stability. This is the first study of adversaries injecting packets into multiple-access channels and restricted this way by individual rates, to the authors' knowledge.

Table 1: A comparison of possibilities and impossibilities to achieve performance characteristics, depending on whether a window adversary of the aggregate injection rate 1 is constrained by a global injection rate or individual injection rates.

class of algorithms	size	quality	global rate	individual rates
ack-based	$n = 2$	stable	I	I
non-adaptive	$n = 2$	bounded latency	P	P
non-adaptive	$n = 3$	stable	I	P
adaptive	$n = 3$	bounded latency	P	P
adaptive	$n \geq 4$	stable	P	P
adaptive	$n \geq 4$	bounded latency	I	P

The parameter n denotes the number of stations. Letter P denotes possibility and I impossibility. The impossibility results hold for channels with collision detection and the possibility results hold for channels without collision detection.

A general goal is to explore what quality of broadcast can be achieved for individual injection rates. In particular, we want to compare adversarial environments defined by individual rates with those determined by global ones, under the maximum average broadcast load of one packet per round. The underlying motivation for this work was that individual injection rates are more realistic in moderate time spans and hopefully the limitations on the quality of broadcast with throughput 1 discovered in [22] could be less severe when the rates are individual. Indeed, we show that bounded packet latency is achievable with individual injection rates when the aggregate rate is 1. This is in contrast with the model of global injection rates, for which achieving bounded waiting times is impossible when the throughput equals 1, as demonstrated in [22].

We denote the number of stations by n and the adversary's window size by w.

The comparison of the two models of global and individual injection rates for a window-type adversary with respect to possibility/impossibility of obtaining stability or bounded packet latency is summarized in Table 1. Additional explanation of this table is as follows. The possibility in a given row means the existence of a broadcast algorithm of a given class (either acknowledgment-based or non-adaptive or adaptive) that always achieves the claimed performance characteristic (either stability or bounded packet latency) in a system of a given size (either $n = 2$ or $n = 3$ or $n \geq 4$). The column for the model of global injection rates reflects the results given in [22].

We further investigate the model of individual injections by deriving bounds on queue size and packet latency. Acknowledgment-based algorithms cannot achieve throughput 1, which strengthens the result for global injection rates [22]. We give a non-adaptive algorithm of $\mathcal{O}(\min(n+w, w \log n))$ packet latency when collision detection is available. An adaptive algorithm can achieve similar performance without collision detection; this is because control bits allow to simulate collision detection with a constant overhead per round. Bounded packet latency can also be achieved by

non-adaptive algorithms in channels without collision detection. More precisely, we develop a non-adaptive algorithm with $\mathcal{O}(n + w)$ size queues and $\mathcal{O}(nw)$ packet latency. The optimality of our non-adaptive algorithm for channels without collision detection, in terms of packet latency, is left open, but we demonstrate optimality in the class of algorithms that avoid collisions altogether.

Previous work. Most of the previous work on dynamic broadcasting in multiple-access channels has been carried out under the assumption that packets were injected subject to stochastic constraints. Such systems can be modeled as Markov chains with stability usually understood as ergodicity, but alternatively, stability may mean that throughput under the given injection rate is as large as the injection rate. Gallager [31] gives an overview of early work in this direction and Chlebus [20] surveys later work. In particular, the popular and early developed broadcast algorithms, like Aloha [1] and binary exponential backoff [43], have been extensively studied for stochastic injection rates. For recent work, see the papers by Bender et al. [15], Goldberg et al. [35, 36], Håstad et al. [38], and Raghavan and Upfal [44]. Stability of backoff algorithms for multiple-access channels with packet injection controlled by a window adversary was studied by Bender et al. [14] in the queue-free model; they showed that the exponential backoff is stable for $\mathcal{O}(1/\log n)$ arrival rates and it is unstable for arrival rates that are $\Omega(\log \log n / \log n)$. Awerbuch et al. [13] developed a randomized algorithm for multiple-access channels competing against an adaptive jamming adversary that achieves a constant throughput for the non-jammed rounds.

Deterministic distributed broadcast algorithms for multiple-access channels in the model with queues were first studied by Chlebus et al. [23] according to the methodology of adversarial queuing. They considered classes of deterministic distributed algorithms, including adaptive and acknowledgment based. They defined latency to be fair when it was $\mathcal{O}(\text{burstiness/rate})$ and considered stability to be strong when queues were $\mathcal{O}(\text{burstiness})$. That paper [23] gave a non-adaptive algorithm achieving fair latency for $\mathcal{O}(1/\text{polylog } n)$ rates and showed that no algorithm could be strongly stable for rates that are $\omega(\frac{1}{\log n})$. They also showed that no oblivious acknowledgment-based algorithm could be stable for rates larger than $\frac{3}{1+\lg n}$, and hence that there are no universally stable oblivious acknowledgment-based algorithms. Two oblivious acknowledgment-based algorithms were developed in [23]: one of fair latency for rates at most $\frac{1}{cn \lg^2 n}$, for a sufficiently large $c > 0$, and an explicit one of fair latency for rates at most $\frac{1}{27n^2 \ln n}$. In the subsequent work on the deterministic distributed algorithms for the multiple access channel with global injection rates, Chlebus et al. [22] investigated the quality of broadcast for the maximum throughput, that is, the maximum rate for which stability is achievable. They defined fairness to mean that each packet is eventually heard on the channel. They developed a stable algorithm with $\mathcal{O}(n^2 + \text{burstiness})$ queues against leaky-bucket adversaries of injection rate 1, which demonstrated that throughput 1 is achievable. They also showed the following inherent limitations on broadcasting with throughput 1: no such algorithm can be fair for a system of at least two nodes against leaky-bucket adversaries, queues may need to be $\Omega(n^2 + \text{burstiness})$, and broadcast algorithms need to be adaptive.

Anantharamu et al. [5] studied packet latency of broadcasting on adversarial multiple access channels by deterministic distributed algorithms when injection rates was less than 1. This was continued by Anantharamu et al. [6] who considered adversarial queuing on multiple-access channels when the adversary can jam the channel.

Anantharamu and Chlebus [4] considered the adversarial model in which the adversary can activate otherwise passive and anonymous nodes by injecting packets into them. The adversary was constrained to be able to activate at most one passive node in a round; such a node remains

active as long as it has packets to broadcast. They showed that positive injection rates could be handled with bounded packet latency; the specific magnitude of rates depends on the class of algorithms, with the injection rate of $\frac{1}{2}$ being the largest one among them. They also demonstrated that no algorithm could achieve bounded packet latency when an injection rate is greater than $\frac{3}{4}$.

Related work. Adversarial queuing has proved to be a viable methodology to represent stability of communication algorithms without statistical assumptions about packet injection. This methodology was first applied to store-and-forward routing. Borodin et al. [19] proposed this in the context of work-preserving routing algorithms when packets are routed along paths stipulated by the adversary at the time of injection, so that routing is restricted to a scheduling policy. The subsequent work by Andrews et al. [9] concentrated on the notion of universal stability, which for a scheduling policy means stability in any network, and for a network denotes stability of an arbitrary scheduling policy, both properties to hold under injection rates smaller than 1. Lotker et al. [42] demonstrated that every work-conserving scheduling policy is stable if the injection rate is suitably small, as determined by the length of the longest path traversed by a packet. FIFO as one of the most popular scheduling policies was studied extensively, see [16, 24]. Rosén and Tsirkin [46] considered routing against rate-1 adversaries; they defined reliability of an algorithm to mean that each packet is eventually delivered and showed that reliability is achievable only in networks with no cycles of length at least 3. Álvarez et al. [2] applied adversarial models to capture phenomena related to routing with varying priorities of packets and to study their impact on universal stability. Andrews and Zhang [10] gave a universal algorithm to control traffic when nodes operate as switches that need to reserve the suitable input and output ports to move a packet from the input port to the respective output one. Álvarez et al. [3] addressed the stability of algorithms in networks with links prone to adversarial failures. Andrews and Zhang [11] proposed suitable adversarial models for networks in which nodes represent switches connecting inputs with outputs so that routed packets encounter additional congestion constrains at nodes when they compete with other packets for input and output ports. Andrews and Zhang [12] studied routing in wireless networks when data arrivals and transmission rates are governed by an adversary. Blesa et al. [18] extended the adversarial model of wired networks to capture variability of speeds of links and sizes of packets.

Static broadcasting in multiple-access channels was considered by Greenberg and Winograd [37], Komlós and Greenberg [40], and more recently by Kowalski [41]. Bieńkowski et al. [17] and Czyżowicz et al. [26] considered algorithmic solutions on multiple-access channels for distributed-computing primitives like consensus and mutual exclusion. Chlebus et al. [21] and Clementi et al. [25] studied the problem of performing independent idempotent tasks by distributed algorithms when processors communicate over a multiple-access channel. The problem of waking up a multiple-access channel was first considered by Gąsieniec et al. [32]; see [27, 28, 39] for later work.

Gilbert et al. [34] proposed to model disruptive interference in multi-channel single-hop networks by a jamming adversary. This was further investigated in a number of papers including the following ones. Dolev et al. [30] considered restricted gossiping in which a constant fraction of rumors needs to be disseminated when the adversary can disrupt one frequency per round. Gilbert et al. [33] studied gossiping in which the adversary can disrupt up to t frequencies per round and eventually all but t nodes learn all but t rumors. Dolev et al. [29] considered synchronization of a multi channel under adversarial jamming.

Document structure. This paper is organized as follows. We summarize and review technical preliminaries in Section 2. Impossibility results and lower bounds are all grouped together in Section 3. The two cases of non-adaptive algorithms for channels with collision detection and adaptive algorithms for channels without collision detection are presented in Section 4. A non-adaptive algorithm for channels without collision detection is given in Section 5. We conclude with final remarks in Section 6.

2   Technical Preliminaries

Multiple access channels are specialized communication networks that support broadcast through their architecture. In this section, we specify what properties of a communication network make it a multiple access channel. Next, we define adversarial models, classes of broadcast algorithms and performance metrics.

We consider networks that operate in a *slotted time*, which means that an execution of an algorithm is a sequence of rounds. All nodes begin and end a round at the same time. These stipulations determine that a network operates synchronously, as if the nodes were coordinated by a global clock.

A node may choose either to transmit or pause in a round. Everything that a node transmits in a round is called *message*. Intuitively, messages overlapping in time collide with each other and none can be successfully received. We assume that messages are big enough so that when two nodes transmit messages in the same round then the transmissions overlap in time.

Multiple access channels. A node receives a feedback from the channel in each round. A message successfully received by a node is *heard* by the node. A broadcast system is a *multiple-access channel* when a message is heard by a node if and only if it is the only message transmitted in this round by any node in the network. This implies that at most one message per round can be heard, so that the throughput rate of a multiple access channel is at most 1. We consider the following three cases determined by the multiplicity of transmissions in a round in order to introduce additional terminology.

> There are no transmissions in a round: nodes receive *silence* from the channel as feed-back. Such a round is called *silent*.
>
> There is one transmission in a round: the message is heard by all the nodes in the same round.
>
> More than one transmissions in a round: no node can hear any message. Multiple transmission in the same round result in a conflict for access to the channel, which we call a *collision*.

Transmitting and listening to the channel are considered independent activities, in that a transmitting node obtains the same feedback from the channel as a node that does not transmit in the round. The channel is *with collision detection* when the feedback from the channel allows the nodes to distinguish between silence and collision, otherwise the channel is *without collision detection*. If no collision detection mechanism is available, then nodes obtain the same feedback from the channel during a collision as during a silent round.

We define a round *void* when no packet is heard in this round. When a round is void then either it is silent or a collision occurs in it.

6

Adversaries. An adversary is defined by a set of allowable patterns of injections of packets into nodes. An adversary generates a number of packets in each round and assigns to each packet the node into which the packet is injected. We first recall the standard definition of globally constrained window-type adversaries. Next we constrain adversaries further by separately specifying how packets are injected into each node.

A (globally constrained) *window-type adversary* is restricted by *injection rate* ρ and *window size* w. These two numbers w and ρ constrain the adversary's behavior as follows: for any contiguous time interval τ of w rounds, the adversary may inject at most ρw packets in τ.

Window adversaries with individual injection rates are defined as follows: Let there be given a positive integer number w, which is again called *window size*. Each node i is assigned a *share* s_i, which is a non-negative integer. The shares satisfy the requirement $\sum_{i=1}^n s_i \leq w$. These numbers constrain the adversary's behavior as follows: in any time interval of w contiguous rounds, the adversary may insert up to s_i packets into node i. The *local injection rate* of node i is defined to be the number $\rho_i = s_i/w$. The *aggregate injection rate* is denoted by $\rho = (\sum_{i=1}^n s_i)/w$. We refer to such a window adversary, with individual injection rates, as being of *local type* $(\langle s_i \rangle_{1 \leq i \leq n}, w)$ and of *aggregate type* (ρ, w). Let a window adversary be of local type $(\langle s_i \rangle_{1 \leq i \leq n}, w)$ with aggregate rate ρ. For this adversary, the *local burstiness at node i* is defined to be its share s_i, and the *aggregate burstiness* is defined to be the number $\delta = \sum_{i=1}^n s_i$, so that $\rho = \delta/w$.

The notion of aggregate type is similar to that of global type as considered in [22, 23].

Broadcast algorithms. We consider communication algorithms that are distributed, in that they are event driven. An algorithm is determined by specifying what constitutes a possible state of a node and what are the rules governing transitions between states.

When a property of a communication environment can be used by algorithms, to be executed in this environment, then we say that this property is *known*. The adversaries we consider are not known, which means that our algorithms are not tailored to any parameters of the adversary.

The letter n denotes the number of nodes attached to a channel, which we may call the *size of the network*. Each node has a unique integer name in the range between 1 and n. We assume that each node knows both its name and the number n.

When multiple packets are pending transmission then some of them need to be parked at nodes waiting to be transmitted. This occurs, for example, when multiple packets are injected simultaneously in the same node. Each node has its packets stored in a private queue. The capacity of such a queue is assumed to be unbounded, in that a queue can store any number of packets in principle.

A message may include at most one packet and possibly some control bits. We do not assume an upper bound on the number of control bits. For example, an algorithm may have nodes attach an "over" bit to a transmitted packet to indicate that the node will not transmit in the next round. A message consisting entirely of control bits is legitimate. The contents of packets do not affect an execution of a broadcasting algorithm, in that packets are treated as abstract markers.

Executions of algorithms. All the nodes start executing a communication algorithm simultaneously. In the course of an execution, the *state of a node* is determined by the values of its private variables, which correspond to the variables used in the code of the algorithm. A node's action in a round of an algorithm's execution consists of the following, in the order given:

7

(1) the node either transmits or pauses, as determined by its state,

(2) the node receives a feedback from the channel, in the form of either hearing a message or collision or silence,

(3) new packets are injected into the node, if any,

(4) a state transition occurs in the node.

An *event* in a round is defined as a vector of nodes' actions in the round, indexed by the nodes. An *execution* of an algorithm is defined as a sequence of events occurring in consecutive rounds. All executions we consider are infinite.

A state transition of a node is determined by the state at the end of the previous round, the packets injected in the round, and the feedback from the channel in the round. Such a transition involves the following operations performed by a node. Any injected packet is enqueued in the same round. A successfully transmitted packet is discarded. When the queue of a node is nonempty, including the packets injected in this round, then the node obtains the next packet to broadcast by dequeuing the queue.

Performance of algorithms. The number of packets stored in a queue in a given round is called the *size of the queue* in the round. An algorithm is *stable in an execution* when there is an upper bound on the sizes of all the queues in this execution. An algorithm is *stable against an adversary* if it is stable in each execution against this adversary. This terminology is naturally extended for classes of adversaries, to mean stability against any adversary in the class. Classes of adversaries are usually determined by properties of their types, like the magnitude of injection rates.

The time spent by a packet waiting in a queue at a station is called this packet's *delay*. An upper bound on packet delay is called packet *latency*. Observe that the size of a queue in a station in a round is a lower bound on some packet's delay, because at most one packet is dequeued in a round. It follows that an algorithm of bounded latency is stable, as latency is an upper bound on queue size.

If an algorithm is stable for a multiple access channel with the first-in first-out (FIFO) queuing discipline in each station, then it is also stable with any other queuing policy. We use the FIFO queuing in each station in each of our algorithms. The reasons are that this discipline minimizes latency and provides fairness even when packet latency is infinite, in the sense that no packet lingers "at the bottom of a queue" forever.

Classes of algorithms. Natural subclasses of deterministic distributed broadcast algorithms in multiple access channels were defined in [22, 23]. We use the same classification, which we give next.

We call an algorithm *adaptive* when control bits can be used in its messages; otherwise an algorithm is called *non-adaptive*.

An algorithm is *acknowledgment based* when a node without packets to transmit stays in an initial state and resets its state to an initial state immediately after a successful transmission.

An acknowledgment-based algorithm is *oblivious* if the decision whether a packet is to be transmitted or not depends only on the station's name and which consecutive round it is devoted to broadcasting a currently handled packet by this station, counting rounds from the first one assigned to this packet.

8

A related terminology is used in the literature to contrast algorithms that sense channel continuously, called "full sensing," with ones that do not sense channel when not having pending packets to broadcast. All algorithms we consider are "full sensing," as our definition of broadcast algorithm stipulates that stations obtain feedback from the channel in each round and make state transitions in each round. The term "acknowledgment-based" is sometimes used to mean that the algorithm has the property that when a station keeps attempting to transmit a packet and eventually succeeds, then the channel "acknowledges" this fact by its feedback, which triggers the station to reset its state to an initial one.

When an oblivious algorithm is executed, then a node may ignore feedback from the channel, except for detecting whether the packet transmitted was heard on the channel, which serves as an "acknowledgment" from the channel. Formally, an oblivious algorithm is determined by unbounded binary sequences assigned to nodes; such a sequence is called a *transmission sequence*. Different nodes, executing the same algorithm concurrently, may have different transmission sequences. These different sequences are interpreted as follows. If the ith bit of the transmission sequence of a node equals 1, then the node transmits the currently processed packet in the ith round, counting rounds from the first one when the packet was started to be processed, while a 0 as the ith bit makes the node pause in the corresponding ith round. Oblivious acknowledgment-based algorithms were considered in [23]. In this paper, acknowledgment-based algorithms are defined by the property that a node begins execution in an initial state and resets its state to initial after a successful transmission; a station without pending packets loops in an initial state.

An algorithm is *conflict free* if in any execution at most one node transmit in any round. An algorithm that is not conflict free is called *conflict prone*. When an algorithm is conflict free then we assume that a channel is without collision detection.

Algorithm design. Now we give an overview of design principles of our algorithms.

It is a natural approach to have nodes work to discover the parameters defining the adversary at hand. The nodes could gradually improve their estimates of the shares s_i and the aggregate maximum burstiness $\delta = \sum_{i=1}^{n} s_i$. The nodes would adjust the frequency of their individual transmissions to the knowledge gained in the process of discovering the adversary. The aggregate burstiness δ is a lower bound on the window w of an adversary. Observe that the aggregate burstiness in the case of injection rate 1 equals the window size.

For a given window adversary of local injection rates, node i is *active* when its share s_i is a positive number; otherwise, when $s_i = 0$, the node i is *passive*. Node i has been *discovered* in the course of an execution when a packet transmitted by i has been heard on the channel. A discovered node is clearly active. In the context of a specific execution, a node that has not been discovered by a given round is called a *candidate* in this round. A passive node is doomed to be a candidate forever. We describe two data structures used to schedule transmissions and have nodes discover their shares.

In one approach, each node i has a private array C_i of n entries. The entry $C_i[j]$ stores an estimate of the share s_j of node j, for $1 \le j \le n$. Every node i will modify the entry $C_i[k]$ in a round in the same way; therefore we may drop the indices and refer to the entries of the arrays as $C[j]$ rather than as $C_i[j]$. The array C is initialized to $C[j] = 0$ for every $1 \le j \le n$. The sum $\gamma = \sum_{i=1}^{n} C[i]$ is an estimate of the aggregate burstiness $\delta = \sum_{i=1}^{n} s_i$.

The nodes running an algorithm keep adjusting the estimates stored in the array C. When node i

enters a state implying $s_i > C[i]$ then i considers itself *underestimated*. Detecting underestimation is implemented by having every node keep track of the transmissions in the past γ rounds. When a node i detects that some $k > C[i]$ packets have been injected within the respective γ consecutive rounds, where k is maximum with this property so far, then i decides in this round that it is *underestimated by the amount $k - C[i]$*.

Another approach to schedule transmissions and discover shares is to have each node i use a list D_i of bits. Such a list D_i has its terms listed as a sequence $\langle d_i(1), d_i(2), \ldots, d_i(\ell) \rangle$. The length ℓ may be modified, but it is the same in all nodes, and additionally the inequality $\ell \leq w$ holds at all times. The lists are initialized to be empty.

The number ℓ will be an estimate on the burstiness of the adversary, similarly as the number γ is for the array C. The number of occurrences of 1 in D_i has the same meaning as $C[i]$ and so is interpreted as the current estimate of the share of i. The lists are maintained so as to satisfy the following invariant: for each $1 \leq j \leq \ell$, when $d_i(j)$ has been determined for all i, then $d_i(j) = 1$ for precisely one i, and $d_k(j) = 0$ for any other value $k \neq i$ such that $1 \leq k \leq n$. When lists D are only used and arrays C are not explicitly maintained, then $C[i]$ will denote the number of occurrences of 1 in D_i. We will use ℓ and γ interchangeably when lists D_i are used.

Our algorithms will have the property that $C[i]$ is increased at most by the amount that node i considers to be underestimated by. This conservative mechanism of estimating the shares is safe, in that the shares are never overestimated, as we show next.

Lemma 1 *If $C[i]$ is increased at most by the amount that node i considers to be underestimated by, then the inequality $C[i] \leq s_i$ holds at all times, for every $1 \leq i \leq n$.*

Proof: We show that the invariant "the inequality $C[i] \leq s_i$ holds for every $1 \leq i \leq n$" is preserved throughout the execution. Initially $C[i] = 0 \leq s_i$, for every $1 \leq i \leq n$, so the invariant holds true in the first round. Observe that if $C[i] \leq s_i$, for every $1 \leq i \leq n$, then also the inequalities

$$\gamma = \sum_{i=1}^{n} C[i] \leq \sum_{i=1}^{n} s_i = \delta \leq w$$

hold true. The adversary can inject *at most* s_k packets into a node k during any segment of γ contiguous rounds, as the inequality $\gamma \leq w$ means that any segment of γ contiguous rounds could be extended to one of w rounds of the execution. A node k can detect to be underestimated by at most $s_k - C[k]$ in any round, while the invariant holds. This makes the invariant still hold in the next round, because an update of $C[k]$ raises $C[k]$ to a value that is at most s_k. $\square$

The algorithms we develop will be such that the assumptions of Lemma 1 hold true, and for them the inequalities $\gamma \leq \delta \leq w$ hold as well. We will use arrays C when each node knows the node for which an update of the current estimate of its share is to be performed. It may happen that such knowledge is lacking: an underestimated node transmits successfully but the other nodes do not know which node transmitted. This is a scenario in which to use lists D, as a transmitting node can append 1 at the end of D while every other node appends 0.

Our algorithms have nodes manipulate auxiliary lists, including the lists D. For each list, there is the *main pointer* associated with this list, which points at an entry we call *current* for the list, when the list is nonempty. In a round, a main pointer either stays put or it is advanced by one position in the cyclic ordering of the entries on the list.

3 Impossibilities and Lower Bounds

In this section, we present impossibility results for acknowledgment-based algorithms and lower bounds on packet latency.

Impossibility results for acknowledgment-based algorithms. We begin by examining acknowledgment-based algorithms. The action that a node performs when it begins processing a new packet is always the same, as it is determined by the initial state. Such an action maybe either to transmit or to pause in the first round.

Lemma 2 *Let us consider an execution of an acknowledgment-based algorithm. If there is a node that pauses in the first round after starting to process a new packet, then, for any number $\rho > \frac{1}{2}$, the algorithm is unstable for some adversary with aggregate rate ρ.*

Proof: Suppose that some node p pauses in the first round of processing a new packet. Consider an adversary who injects only into such a node p as often as possible subject to an individual injection rate that is between $\frac{1}{2}$ and ρ. This results in an execution in which a packet is heard not more often than every second round, while the aggregate rate is greater than $\frac{1}{2}$, so the queue at p grows unbounded. $\qquad\square$

Theorem 1 *Any acknowledgment-based algorithm is unstable in the multiple-access channel with collision detection that consists of just two nodes, when executed against some window adversary of burstiness 2 and aggregate injection rate 1.*

Proof: Consider an acknowledgment-based algorithm for two nodes p and q. Suppose, to arrive at a contradiction, that the algorithm is stable for the aggregate injection rate 1. This implies, by Lemma 2, that a node transmits a new packet immediately.

We define an execution of the algorithm with an infinite sequence of rounds $t_1, t_2, \ldots$ determined so that there are at least i void rounds by round t_i. (Recall that a round is void when no packet is heard in this round.) The adversary will inject two packets in each odd-numbered round, a packet per node, and it will inject no packets in even-numbered rounds. This means that each node has its individual injection rate equal to $\frac{1}{2}$. The aggregate injection rate of the adversary is 1, and so there will be at least i packets in queues in round t_i. Next we stipulate how such a scenario can occur.

Let us set t_1 to be the first round. This round is silent as the nodes did not have any packets in round zero. A collision occurs in the second round. This is because each node got a new packet in the first round, so it transmitted it immediately. Let us define t_2 to be the second round. As our construction continues, each round number t_i, for $i > 1$, will be even. The adversary will not inject packets in these specific rounds t_i, as packets will not be injected in any even-numbered round, as a general rule.

Suppose the execution has been determined through an even-numbered round t_i. If t_i+1 is void then define $t_{i+1} = t_i + 2$. Otherwise some node, say p, transmits in round $t_i + 1$. The node p will continue transmitting for as long as it has packets, because it transmits a new packet immediately. If in one such a round t the node q transmits concurrently with p, then this results in a collision and we define t_{i+1} to be t if t is even or $t + 1$ otherwise. Let us suppose that this is not the case,

so p keeps transmitting alone. The node p will not have a packet to transmit in some round after $t_i + 1$, since its injection rate is $\frac{1}{2}$ and so smaller than 1; let $t' > t_i + 1$ be the first such a round. Observe that t' has to be an odd number, as otherwise a new packet would have got injected into p in the round $t' - 1$ resulting in p having a packet to transmit in round t'. If q does not transmit in round t', then this round t' is silent and we are done; in this case we define t_{i+1} to be $t' + 1$, as such a round number is to be even. Otherwise, the node q transmits in round t' successfully. Simultaneously both nodes obtain new packets in round t', so each has at least one available packet at the end of this round. Each node transmits in round $t' + 1$, as for each of them it is a first round of processing a new packet. This is because p did not have a packet in round t' and q transmitted in round t'. Define t_{i+1} to be the void round $t' + 1$. This completes the inductive construction of the sequence t_i, and by this produces a contradiction with the assumption that the algorithm is stable. $\qquad\square$

Next we investigate how large could an injection rate be, as a function of the number of nodes attached to a multiple access channel, that can be handled in a stable manner by an acknowledgment-based algorithm. It was shown by Chlebus et al. [23] (Theorem 5.1) that an oblivious acknowledgment-based algorithm cannot be stable when the global injection rate is at least as large as $\frac{3}{1+\lg n}$, for $n \geq 4$ nodes, where $\lg n$ denotes the binary logarithm of n. We show a related result for individual injection rates and acknowledgment-based algorithms that are not necessarily oblivious. The following theorem was inspired by the related result in [23] and its proof is similar to that given in [23].

Theorem 2 *If an acknowledgment-based algorithm is executed by $n \geq 4$ nodes on a multiple-access channel with collision detection, then the system is unstable against an adversary for which only two nodes have positive shares, one such share is equal to 1 and the other to 2, and the window size is larger than $\lceil \lg n \rceil$.*

Proof: Let $\mathcal{A}$ be a specific acknowledgment-based algorithm for the n nodes. For each node p, consider an execution of algorithm $\mathcal{A}$ when p starts to work on a new packet. Let us consider an execution, understood as an experiment, such that when p transmits then p hears a collision and when p does not transmit then the round is silent. The purpose of this experiment is to determine a sequence of bits $s(p) = (b_1, b_2, \ldots)$ such that $b_i = 1$ when p transmits in the ith round and $b_i = 0$ when p pauses in the ith round in this execution, where we count rounds from the first one when p begins to process the packet. Let $s(p, i)$ be this sequence truncated to the first i terms, for $i \geq 1$. There exist two nodes, p_1 and p_2, for which $s(p_1, \lceil \lg n \rceil - 1) = s(p_2, \lceil \lg n \rceil - 1)$ by the pigeonhole principle, because $\lceil \lg n \rceil - 1 \geq 1$ for $n \geq 4$ and $2^{\lceil \lg n \rceil - 1} = 2^{\lceil \lg n \rceil}/2 < n$.

If the two sequences $s(p_1)$ and $s(p_2)$ are identical then it is sufficient for the adversary to simultaneously inject one packet into p_1 and another into p_2 and these packets will never be heard on the channel. Then instability follows, because packets subsequently injected into p_1 and p_2 will need to be queued and none of them will ever be heard. Otherwise, let $k \geq \lceil \lg n \rceil$ be the first position in which the sequences $s(p_1)$ and $s(p_2)$ differ. Without loss of generality, suppose that the kth term of $s(p_1)$ is 1 and the corresponding term of $s(p_2)$ is 0. Let j be the smallest position of the sequence $s(p_2)$ such that $j > k$ and there is 1 at this position in $s(p_2)$. We determine an adversary as follows. The window size $w = j$ and node p_1 has share 1 and node p_2 has share 2, which is possible for $n \geq 4$ because then $j \geq 3$.

This adversary may inject the packets as follows. At round number 0, the adversary injects into

each node the number of packets equal to that node's share. This is followed by w rounds so that two packets are successfully transmitted, one in round k by node p_1 and the other in round $w = j$ by node p_2, while node p_2 still has one packet. At round $w = j$, the adversary injects into each node the number of packets equal to this node's share. This makes the behavior of the channel during the rounds $w + 1$ through $2w$ be the same as during the rounds from 1 through w, with the only difference that at round $2w$ node p_2 has two packets pending transmissions. This pattern can be iterated forever, with the result that at the round number ℓw node p_2 has ℓ pending packets. $\square$

Due to the inherent limitations of acknowledgment-based algorithms, as expressed in Theorems 1 and 2, the algorithms we develop in this paper are not acknowledgment based.

Lower bounds. Next we present two lower bounds on packet latency. We begin by showing that an algorithm with performance close to optimal needs to have bounds $\Omega(n + w)$ or $\Omega\big(w \operatorname{polylog} n\big)$ on packet latency.

Theorem 3 *For any broadcast algorithm for a channel with n nodes and for an adversary of window w and aggregate injection rate 1, packet latency provided by the algorithm in some execution is $\Omega(w \max(1, \log_w n))$, when $w \le n$, and it is $\Omega(w)$, when $w > n$.*

Proof: The adversary can inject w packets in one round because the aggregate injection rate is 1 so the burstiness equals w.

For the case $w \le n$, we consider adversaries for which each node has a share that is either 0 or 1. Greenberg and Winograd [37] considered a static version of the broadcast problem, in which k packets are located initially among k nodes, for some $k \le n$, at most one packet per node, and the goal is to have all of them heard on the channel. They showed that for any algorithm it takes $\Omega\big(k \frac{\log n}{\log k}\big)$ time to achieve this goal in some execution of the algorithm. Algorithms that we use handle dynamic broadcast, but they can be applied to the static version. Namely, a translation to the static version of broadcast is as follows: let the adversary inject w packets in the first round, at most one packet per node, the algorithm needs to broadcast only these packets. This gives the bound $\Omega\big(w \frac{\log n}{\log w}\big) = \Omega(w \max(1, \log_w n))$.

Next consider the case $w > n$. Let the adversary inject w packets in the first round and then execute the algorithm. It will take at least w rounds to hear these packets. $\square$

The next observation is that to attain latency $o(nw)$, the algorithm has to be conflict prone.

Theorem 4 *For any conflict free algorithm, a system of sufficiently many nodes n, and any sufficiently large window w, there is an execution in which a packet is delayed by $\Omega(nw)$ rounds against a window adversary with window w and an aggregate injection rate smaller than 1.*

Proof: Let us consider a specific conflict-free algorithm and its execution. The adversary is specified in two stages. The first stage is by declaring some node to be *heavy* and the remaining ones to be *mavericks*. The share of the heavy node is $w - 2$, and one of the mavericks has its share equal to 1, so that the aggregate injection rate is $\frac{w-1}{w} = 1 - \frac{1}{w} < 1$. At this point, the heavy node may be "known to the algorithm" while the maverick with a positive share will be declared in the second stage, some time later in the execution, depending on the algorithm's actions.

Consider an execution $\mathcal{E}_1$ in which the adversary injects only into the heavy node, with full capacity of $w - 2$ packets per w consecutive rounds, and does not inject into any maverick node

13

at all. Let us partition any execution into disjoint segments of consecutive $\frac{(n-1)w}{2}$ rounds. The adversary injects $\frac{(n-1)(w-2)}{2}$ packets into the heavy node during each segment. This leaves a room for $n-1$ rounds during a segment of $\mathcal{E}_1$ that are available to locate the maverick with a positive share. We argue that an algorithm cannot locate such a maverick without incurring $\Omega(nw)$ delay.

We specify an execution $\mathcal{E}_2$ which has the same prefix as $\mathcal{E}_1$ until the first injection into a maverick. To this end, we consider the consecutive segments of $\mathcal{E}_1$ one by one, starting from the first segment; let S denote a current segment of $\mathcal{E}_1$. We proceed through a sequence of cases, depending on how many mavericks are scheduled to transmit in S.

If fewer than $n-1$ mavericks are scheduled to transmit in S, then some maverick is not scheduled to transmit in S at all. We switch to $\mathcal{E}_2$ by having the adversary inject a packet into a maverick that is not scheduled to transmit in S, the injection occurring at the round just before segment S is to begin. This packet waits at least the duration of a segment, which is $\Omega(nw)$. If such a segment S exists then the argument is completed.

Let us suppose that, alternatively, every maverick is scheduled to transmit at least once during every segment of $\mathcal{E}_1$. If some maverick is scheduled to transmit more than once in S, then the number of packets in the heavy node increases during this segment S, because at least one time slot contributes to delaying the heavy node in unloading its packets; in such a case we proceed to the next segment of $\mathcal{E}_1$. If only such segments are in $\mathcal{E}_1$, then the algorithm is not stable and packet delays are arbitrarily large. Otherwise, suppose that there is a segment S during which every maverick is scheduled to transmit exactly once. Let us partition S into first and second halves, each of $\frac{(n-1)w}{4}$ rounds. We consider two sub cases next.

The first sub-case occurs when the last maverick to transmit in S is scheduled to transmit in the first half of S. We switch to $\mathcal{E}_2$ by having the adversary inject into this last maverick just after its scheduled transmission in S. This packet needs to wait at least for the duration of the second half, which is $\Omega(nw)$.

The other sub-case occurs when the last maverick to transmit in S is scheduled to transmit in the second half of S. We switch to $\mathcal{E}_2$ by having the adversary inject into this last maverick just before segment S is to begin. This packet needs to wait at least for the duration of the first half, which is $\Omega(nw)$.

The considered cases and sub-cases exhaust all the possibilities, and we conclude that a packet's delay $\Omega(nw)$ may occur for the considered adversary and injection rate $1 - \frac{1}{w}$. $\qquad\square$

In Section 5 we give a collision-free non-adaptive algorithm for channels without collision detection that has $\mathcal{O}(nw)$ packet latency. The question if packet latency has to be $\Omega(nw)$ when a collision-free non-adaptive algorithm is executed on channels without collision detection is left open.

4   Two Algorithms of Small Latency

In this section we present algorithms with packet latencies that are close to optimal against an adversary of aggregate injection rate 1. The algorithms are developed for the two scenarios when either (1) collision detection *is* available, in which case we do not use control bits in messages, or (2) collision detection is *not* available, in which case we do use control bits in messages.

4.1 A non-adaptive algorithm with collision detection

We develop a non-adaptive algorithm Non-Adaptive-Discover-Shares which uses collision detection to bound latency. Algorithm Non-Adaptive-Discover-Shares in turn uses two algorithms Search-Collision-Update and Cycle-Collision-Update executed in sequence. Algorithm Search-Collision-Update is executed first and next we switch to Cycle-Collision-Update after a certain number of collisions is reached in an execution of Search-Collision-Update.

A node i uses the list of bits D_i to implement its estimate of shares. The nodes manipulate the lists according to the principles given at the end of Section 2, we recall them next. Each such a list D is initially empty. There is a (main) pointer associated with each list, which points at the current entry. The pointer is set to the first position on a list, when the first entry is inserted into the list, and after that it is advanced in the circular order of the list by one position in each round.

The lists D are used to schedule transmissions as follows: a node i transmits in a round when its current entry in D_i is 1 and otherwise it pauses. A node i that upgrades its share appends an entry with 1 to the end of its list D_i, while at the same time all the other nodes append 0's to their lists. This provides conflict freeness, because the following invariants about lists D_i, for $1 \le i \le n$ are maintained in each round:

(i) the lists at the nodes are all of the same length,

(ii) the main pointers of all lists are on the same position in their respective lists,

(iii) at most one main pointer indicates a current entry with 1 while all the remaining pointers indicate at 0's.

Each of algorithms Search-Collision-Update and Cycle-Collision-Update has two threads, the main one and the update one, but they are implemented and cooperate differently.

Algorithm Search-Collision-Update. Now we explain in detail how algorithm Search-Collision-Update works. The algorithm starts by invoking the main thread. The main thread occasionally calls the update thread, when it is needed. The update thread uses a binary search among the nodes to locate processors that consider themselves underestimated. Next we describe the threads in detail.

The main thread uses the lists D. This prevents conflicts for access to the channel, as long as there are no transmissions beyond those schedule by the lists D. While the main thread is executed, some nodes may detect that there are underestimated. Any such an underestimated node becomes *persistent*, which means it will work to create a collision in order to gain an opportunity to upgrade its share. A persistent node transmits in each round, even when it is not scheduled to transmit by its array D, as long as it has packets. If a collision occurs, then the update thread is invoked, while the main thread pauses. This means that the main pointers associated with the lists D are not advanced and persistent nodes do not transmit, except as participants of the binary search performed by the update thread. The main thread resumes after the update thread terminates.

The update thread operates as follows. It performs search over the range of nodes by referring to intervals of names of nodes. In each round one interval is distinguished as *current*. In a round, each node in the current interval that wants to upgrade its share transmits a pending packet. The search

15

starts with the interval $[1, n]$, comprising all the names, which is set as current. If a collision is heard then the current interval is partitioned in two subintervals in a balanced way. These intervals will be processed recursively using a stack. The left interval is pursued first, by becoming current, while the right interval is pushed on the stack. If a silence is heard for some current interval then such an interval is abandoned and the next interval is popped from the stack and made current. If a packet is heard then the node that transmitted the packet is considered as having reserved the channel to transmits a number of packets up to its share's upgrade, which then is followed by silence. Such a transmitting node appends an entry with 1 to its list D_i for each transmission of a packet during upgrading its share, while the other nodes append 0's each. The silence after a sequence of transmissions makes all stations abandon the currently processed interval and the next interval is popped from the stack and made current. The update thread terminates after the stack becomes empty and the main thread resumes.

This completes the specification of algorithm SEARCH-COLLISION-UPDATE. Let us recall the notation $\gamma = \sum_{i=1}^{n} C[i]$ interpreted as an estimate of the aggregate burstiness $\delta = \sum_{i=1}^{n} s_i$, as it was introduced in Section 2. In what follows, when we refer to γ then we mean the ultimate value it attains in an execution, like in the formulations of Lemmas 3 and 4, unless indicated otherwise, like, for example, in the specification of algorithm NON-ADAPTIVE-DISCOVER-SHARES.

Lemma 3 *Packet latency of* SEARCH-COLLISION-UPDATE *is* $\mathcal{O}(\gamma(1 + \log n))$ *when it is executed in a system of n nodes.*

Proof: We observe that once all the upgrades in an execution have been performed, packet latency is proportional to the maximum number of packets queued simultaneously plus burstiness. The burstiness experienced in an execution is at most (the final value of) γ. Each upgrade of a share involves the upgrade thread, which takes at most $\lceil 1 + \lg n \rceil$ rounds per upgrade. There are at most γ invocations of the upgrade thread, so the total number of rounds spent on upgrades is at most $\lceil \gamma(1 + \lg n) \rceil$. The number of packets injected during the rounds spent on upgrades is at most the number of these rounds plus burstiness δ, which is $\mathcal{O}(\gamma(1 + \log n))$. It follows that the maximum number of packets queued simultaneously is $\mathcal{O}(\gamma(1 + \log n))$, and the packet latency is also $\mathcal{O}(\gamma(1 + \log n))$. $\square$

Algorithm Cycle-Collision-Update. Now we explain in detail how algorithm CYCLE-COLLISION-UPDATE operates. The two threads, main and update, start simultaneously. They work concurrently, unless stated otherwise, for example when the main thread is paused for upgrading shares.

The main thread uses the lists D, similarly as the main thread of algorithm SEARCH-COLLISION-UPDATE. This means that a node i that has a 1 as a current entry in its list D_i in a round transmits if it has a packet. The update thread uses a separate list of the names of all the nodes ordered in a circular order. It is represented by a copy in each station, with a main pointer associated with it. A node that is current on this list is referred to as *current for update.* A node transmits in this round when the following is satisfied: the node is current for update, it considers itself underestimated, and it has a pending packet.

The feedback from the channel after a transmission by a node current for update can be of two kinds: either a collision or a packet heard. We consider each of them next.

If the message is heard then the transmitting node turns *persistent*, which means in will transmit

a packet in every subsequent round, as long as it has pending packets. Simultaneously, the main pointer on the circular list is advanced so that the next node becomes current for update; such an advance of this pointer occurs in each round until a collision is heard. A node that is persistent turns back to non-persistent when either its queue becomes empty or when a collision occurs or when the node becomes current for update again.

Notice that there is only one persistent node at any time. This is because when a node becomes persistent after a successful transmission, it continues to transmit until either it exhausts its packets or a collision occurs, which terminate the property of being persistent. In particular, no other node can become simultaneously persistent because this would mean two simultaneous successful transmissions: one by the first persistent node and another one by a contender node.

Now we discuss the case of a collision, which is caused by either two or three concurrent transmissions. This bound on the number of concurrent transmissions is the case because only the following cases of transmissions are possible. First occurs when a transmitter executes the main thread, which means the transmission is determined by the D list. Second occurs when a transmitter is ready for update. Third occurs when a transmitter is persistent.

When a collision is heard then the main thread pauses and the update threads proceeds to upgrade shares. First a possible transmitter that is ready for update is given a chance to transmit. If it wants to upgrade its share then it transmits a number of times, up to its share's upgrade, followed by a silent round. Next a possible persistent transmitter is given a chance to upgrade its share. This is performed similarly, by having it transmit a number of times, up to its share's upgrade, followed by a silent round. These two upgrades are performed by appending 1's to the list D of the transmitting node and 0's to such lists of the other nodes, an entry for each transmission of a packet. Such two silent rounds occur eventually. The first indicates a completion of upgrade by a transmitter that is ready for update, if there is any. The other indicating a completion of upgrade by a persistent node, if there is any. After the second of these two silent rounds occurs, both the main thread and the update one resume regular operations, by having pointers on their respective lists advance in each round.

Lemma 4 *Packet latency of algorithm* CYCLE-COLLISION-UPDATE *is* $\mathcal{O}(n + q + \gamma)$ *when the algorithm is executed in a system of n nodes with q packets in queues at the start.*

Proof: A packet delay may be attributed to either to the number of packets inherited in the queue or to the time spent waiting for an upgrade of a node, in which a packet resides and that considers itself underestimated, or to the time spent to upgrade shares. There are q packets inherited in queues. The time waiting to start an upgrade is at most n, because the list used by the upgrade thread consists of n entries. An update costs at most three void rounds, as they comprise one collision and two silences. These void rounds for each share's upgrade happen only once. As there are γ upgrades, the total packet delay is as claimed. □

The ultimate algorithm Non-Adaptive-Discover-Shares. Algorithm NON-ADAPTIVE-DIS-COVER-SHARES starts by invoking algorithm SEARCH-COLLISION-UPDATE. The estimate γ of the aggregate burstiness is available in each round, as explained in Section 2. As long as the inequality $n + \gamma \geq \gamma(1 + \lg n)$ holds, where γ is understood as changing in time, then algorithm SEARCH-COLLISION-UPDATE is executed. When the inequality $n + \gamma < \gamma(1 + \lg n)$ starts to hold then algorithm SEARCH-COLLISION-UPDATE stops and algorithm CYCLE-COLLISION-UPDATE takes over. The

algorithm is well specified because the inequality $n + \gamma \geq \gamma(1 + \lg n)$ holds for small values of γ, for example for $\gamma = 0$, and the inequality $n + \gamma < \gamma(1 + \lg n)$ holds for sufficiently large γ, for example for $\gamma \geq n$.

Theorem 5 *Algorithm* NON-ADAPTIVE-DISCOVER-SHARES *provides* $\mathcal{O}(\min(n + w, w(1 + \log n)))$ *packet latency for a channel with collision detection against the adversary of window w and aggregate injection rate 1 in a system of n nodes.*

Proof: If algorithm CYCLE-COLLISION-UPDATE is not invoked, then $n + \gamma \geq \gamma(1 + \lg n)$ holds in all rounds. In this case, packet latency is $\mathcal{O}(\gamma(1 + \log n))$ by Lemma 3.

When CYCLE-COLLISION-UPDATE is invoked, then $n + \gamma < \gamma(1 + \lg n)$ holds, when γ is understood to denote the value of this parameter at the round of invocation of CYCLE-COLLISION-UPDATE. This is the first round when the inequality holds, so then $\gamma = \Theta(n/\log n)$. There are $\mathcal{O}(n + \gamma) = \mathcal{O}(n)$ packets in queues in a round of invocation of CYCLE-COLLISION-UPDATE, because the time spent on upgrades is $\mathcal{O}(\gamma(1 + \lg n))$ and the experienced burstiness is $\mathcal{O}(\gamma)$. In this case, packet latency is $\mathcal{O}(n + q + \gamma)$, by Lemma 4, where $q = \mathcal{O}(n)$ is the number of packets in queues at the start of CYCLE-COLLISION-UPDATE. We obtain that $\mathcal{O}(n + \gamma)$ is an upper bound on packet latency.

It follows that algorithm NON-ADAPTIVE-DISCOVER-SHARES provides $\mathcal{O}(\min(n + \gamma, \gamma \log n))$ packet latency. The number γ depends on an execution, but the inequality $\gamma \leq w$ holds, by Lemma 1. We observe that this implies $\min(n + \gamma, \gamma(1 + \log n)) \leq \min(n + w, w(1 + \log n))$. This can be shown considering the following three cases. If $n + w \geq w(1 + \log n)$ then $n + \gamma \geq \gamma(1 + \lg n)$ and $w(1 + \lg n) \geq \gamma(1 + \lg n)$. If $n + w < w(1 + \log n)$ and also $n + \gamma < \gamma(1 + \lg n)$ then $n + \gamma > n + w$. If $n + w < w(1 + \log n)$ but $n + \gamma \geq \gamma(1 + \lg n)$ then we note that $\gamma(1 + \lg n) \leq n + \gamma \leq n + w$. □

4.2 An adaptive algorithm without collision detection

Now we give an adaptive algorithm for channels in which collision detection is not available. We call the algorithm ADAPTIVE-DISCOVER-SHARES. This algorithm simulates the two algorithms in NON-ADAPTIVE-DISCOVER-SHARES by way of running algorithms SEARCH-SILENCE-UPDATE and CYCLE-SILENCE-UPDATE. The simulation proceeds as follows.

In both simulations of the corresponding main threads, a node scheduled to transmit but without pending packets in its queue transmits a control bit to indicate this, rather than pause and produce a silent round. It follows that a silence occurs only when a thread modified in this way creates a collision. Such a silent round results in invoking the update thread.

The update thread in SEARCH-COLLISION-UPDATE relies on collision detection to implement a binary search. Now we need to detect a collision among silent rounds. Silence heard in a round t during the update thread indicates that either there was no transmissions or a collision occurred. We resolve which of these is the case in the next $t + 1$-st round as follows. All the nodes that transmitted in round t transmit together with node 1. Node 1 may have transmitted in round t, but it transmits a message with a control bit in round $t + 1$, so it does not need to have a pending packet. There are two possible events occurring in round $t + 1$: either the round is silent or a message is heard. A silence in round $t + 1$ indicates that more than one node transmitted, as node 1 certainly did transmit, so there was at least one node transmitting in the previous round t. This

18

means that there was a collision in round t, as no message was heard in that round. If a message is heard in round $t + 1$, then this only can be the message transmitted by node 1. Therefore no other node transmitted in the previous round t, and so round t was silent.

The simulation of CYCLE-SILENCE-UPDATE proceeds as follows. When the simulated main thread is running, then, in each round, some node transmits a message, as prompted by the occurrence of 1 in its list D. Therefore a silent round indicates a collision. When this occurs, some two nodes (a candidate for update and a persistent one) are given an opportunity to upgrade their shares. This is performed by a sequence of transmissions in consecutive rounds, each resulting in a message heard. Each among these two nodes, if any, indicates a completion of this task by a silent round (when there is no corresponding node then a silent round indicates that). Therefore two silent rounds in this situation indicate lack of transmissions in them, which triggers the two threads to resume advancing pointers on their lists.

Theorem 6 *Algorithm* ADAPTIVE-DISCOVER-SHARES *provides* $\mathcal{O}(\min(n + w, w \log n))$ *packet latency for a channel without collision detection against an adversary of window w and aggregate injection rate 1 in a system of n nodes.*

Proof: The simulation we employ produces a constant overhead per each simulated round, which is verified by a direct inspection of the simulation mechanism. Therefore packet latency of the simulating algorithms is of the same order of magnitude as that of the simulated algorithm. Theorem 5 gives a bound for the simulated algorithm, and the same asymptotic bound holds true for the simulating algorithm. □

5   A Non-adaptive Algorithm without Collision Detection

In this section we consider channels without collision detection. We develop a non-adaptive algorithm of bounded packet latency for aggregate injection rate 1. The algorithm is called COLORFUL-NODES. The algorithm has each node maintain a private list of names of discovered active nodes. The name of a newly discovered node is apparent to each node so it is immediately appended to the lists. There is a pointer associated with the list of names of discovered nodes, which is occasionally advanced by one position in a circular manner; when this happens then all the nodes do this in unison. It follows that these lists are identical and are manipulated in the same way by each node

An execution of algorithm COLORFUL-NODES begins with a stage we call preparation, which is followed by phases that are iterated in an unbounded loop. A *phase* consists of three consecutive stages. A pure stage occurs first in a phase, it is followed by an update, and finally a makeup concludes the phase. It may happen in some phase that the update and makeup stages are missing, to the effect that the initial pure stage of the phase comprises the whole remaining part of the execution. Such a situation may occur only when, starting from a certain point in time in this pure stage, a packet is heard in every round.

Intuitions and motivation behind stage designs. The following are the intuitions that have motivated and guided the design of stages in algorithm COLORFUL-NODES.

The purpose of the preparation stage is merely to discover at least one active node. This stage occurs only once.

Pure stages are for transmissions by the active nodes that are already discovered. The amount of time allotted for such transmissions is determined by the bounds on shares of the nodes as they have been estimated up to this stage. It is during pure stages that most of the work of broadcasting is expected to be accomplished.

An update stage serves the purpose to give nodes an opportunity to announce to be underestimated. Such announcements result in the respective entries of the array C getting incremented in order to improve the current estimates of shares. Nodes that are not underestimated pause during an update stage, so these rounds could be considered wasted for them. Eventually, no node is underestimated in an execution. After this happens, update stages consist of silent rounds only, if any occur.

In any scenario, an update stage includes n silent rounds. We do not rush into an update stage when a pure stage is under way; we wait until n silent rounds have been accrued during a pure stage. Each such a silent round indicates that the corresponding node scheduled to transmit has no packets.

When a pure stage ends, then the nodes are partitioned into those that have been detected to have no packets at some point of the last pure stage and the remaining "busy" ones that have not been detected as such. The silent rounds in pure and update stages create a potential for the queues to grow unbounded at nodes that consistently obtain their maximum load of packets. It is the purpose of a makeup stage to compensate "busy" nodes for the rounds wasted during the forced silent rounds of the preceding pure and update stages.

Details of stage implementation. We describe the details of implementation for each kind of stage: preparation, pure, update, and makeup.

The *preparation* stage is organized such that every node has one round to transmit, assigned in a round robin manner. A node with a packet available transmits one when the node's turn comes up, otherwise the node pauses during its time slot. The preparation terminates after some node performs the first transmission. The node i whose packet has been heard during the preparation becomes discovered, which is recorded by setting $C[i] \leftarrow 1$.

During a *pure* stage, the discovered nodes proceed through a sequence of transmissions, starting from the current node on the list of discovered nodes. A node i has a segment of consecutive $C[i] > 0$ rounds allotted for exclusive transmissions. During this segment of rounds, the node i keeps transmitting as long as it has packets, otherwise the node i pauses. The pointer is advanced, and the next node takes over, when either the current node i has used up the whole segment of $C[i]$ assigned rounds or just after node i did not transmit while scheduled to. After n silences occur during a pure stage, then this concludes the stage and an update stage follows.

During a pure stage, a *marker* is generated each time a silent round occurs. Markers come with one of two colors. When a node i holds a *green marker*, then this color indicates that the marker was generated when i was silent during a round in a segment of $C[i]$ rounds allotted for i to transmit. A *red marker* held by node i indicates that it was some other node j, for $i \neq j$, that was silent during a round in a segment of $C[j]$ rounds allotted to j to transmit, which generated the marker. Every node keeps a list of markers and their assignments to nodes in its private memory. All nodes perform operations on markers in exactly the same way in unison. No node holds a marker in the beginning of a pure stage. Only nodes already discovered get markers assigned to them during a pure stage. A discovered node may hold either no markers or a green one or a red

one in a round of a pure stage.

All operations on markers are triggered by silences during update stages. When a new marker is created and assigned to a node then some old markers may be reassigned. The details are as follows.

Let a node i be silent in a round assigned to i to transmit in an update stage: this generates a marker.

1. If i does not hold a marker yet, then the new marker is colored green and it is assigned to i.

2. If i already holds a green marker, then the new marker is colored red and it is assigned to the first available discovered node, in the order of their names, that does not hold a marker yet.

3. If i holds a red marker, then the new marker is colored green, it is assigned to i, while the original red marker held by i is reassigned to a discovered node that does not hold a marker yet.

A pure stage terminates after every discovered node gains a marker. A discovered node is considered *colored* by the same color as the marker it holds when an update begins. We need markers only to assign colors to discovered nodes. After every discovered node has gained a marker, and hence a color, then we refer only to the nodes' colors. Colors remain assigned to the discovered nodes through the end of the next makeup stage, and so to the end of the phase. Some colors will be modified during the makeup stage.

An *update* stage is to give every node one opportunity to transmit exclusively for a contiguous segment of rounds. This does include candidate nodes. A node i that is underestimated by an amount x transmits x times, which is followed by a silent round. (It might happen that an underestimated node does not have sufficiently many packets ready to be used to announce by how much it is underestimated in an update stage. This does not create a design issue, as this indicates that so far the node has had enough room to transmit its packets.) A number $y \le x$ of transmissions in such a situation results in an immediate increment $C[i] \leftarrow C[i] + y$ at all nodes. In particular, when a new node k becomes discovered, then this results in setting $C[k]$ to a positive value. When a node simply pauses in the first round assigned to it, then the corresponding entry in the array C is not modified. In particular, when a candidate node j does not transmit then it maintains its candidate status, which is represented by $C[j] = 0$. After each node has had a chance to perform all its transmissions in a update stage, then the stage terminates and a makeup stage follows next.

Green nodes have had their queues empty at some point in the last pure stage. A *makeup* stage has a purpose to have red nodes empty their queues as well. These nodes transmit in the order inherited from the list of discovered nodes, starting from the current node, if it is red, or otherwise the next red one following the current node on the list. A red node i has a segment of consecutive $C[i]$ rounds allotted for exclusive transmissions each time it becomes current while red. After a red node i performs $C[i]$ transmissions then it maintains the red status but stops being current, unless it is the only red node in a round. A silent round by a red node i, during $C[i]$ assigned rounds, results in changing the color of the node to green immediately and advancing the pointer to the next red node, if any. A makeup stage concludes as soon as there are no more red nodes.

This concludes the specification of all kinds of stages, and hence of algorithm COLORFUL-NODES.

Observe that algorithm COLORFUL-NODES is conflict free. This follows by the design of stages.

The relevant property is that each stage uses a list of nodes and a pointer on this list indicates which node is to transmit in a given round.

The performance of algorithm Colorful-Nodes. The algorithm uses four kinds of stages. Assuming that some packets are injected, three stages are of bounded duration while one is not necessarily so. A preparation stage ends as soon as a packet is heard, which is a certain event assuming that some packets are injected. An update stage takes up to $n + w$ rounds. A makeup stage is also of bounded duration, as we will show in Lemma 7. Pure stages are different in that it is possible that some pure stage does not end. If this is the case, then there are finitely many phases, with the last one ending in a pure stage. In such a scenario, starting from some round, a packet is heard in every round.

Lemma 5 *A packet stays in a queue during at most two consecutive phases.*

Proof: Let us consider a packet injected into some node v in a phase J. If either phase J or phase $J + 1$ are the last phases then the packet cannot stay beyond them.

In what follows, we assume that both phases J and $J + 1$ end, which means that each stage in them ends. This implies that the queue of each node becomes empty at some point during each phase. This property is provided by the design of the algorithm and by the meaning of colors of nodes. Green nodes get their queues empty early in a phase, during the respective pure stage, which comes to an end by the assumption in this case. Red nodes accomplish getting their queues empty during the next makeup stage, which also comes to an end by the same assumption. A packet is heard by the first round following its injection in which its queue becomes empty.

If the considered packet gets injected before the queue at v becomes empty during phase J, then all the rounds when the packet stays in its queue are included in phase J. When the queue of v gets empty in the phase J before the packet is injected in this very phase J, then the packet will be heard by the end of the next phase $J + 1$ at the latest, by the round in which the v's queue gets empty in phase $J + 1$. In any case, a packet injected in phase J gets heard by the end of phase $J + 1$. $\qquad\square$

At most n silent rounds occur during the preparation stage, starting from the first packet injection. Similarly, there are at most n silent rounds during every stage that follows. For accounting purposes, we partition silent rounds into *blocks* defined as follows. The first block comprises the silent rounds that occur during the last n rounds of the preparation stage (or all the silent rounds during the preparation stage, in case there are less than n rounds in the preparation stage), and those silent rounds that occur during the first pure stage and, assuming that this pure stage ends, the silent rounds of the first update stage. The next block, if it exists, consists of at most $3n$ silences incurred during the immediately following makeup, pure and update stages, assuming that all these stages exist. This continues throughout an execution, a block comprising silences in consecutive makeup, pure and update stages. When some stage does not end, then this results in some block occurring last and having fewer silent rounds than it would have otherwise.

Lemma 6 *There are $\mathcal{O}(n + w)$ packets in queues in any round of an execution of algorithm* Colorful-Nodes *against adversaries of window w and aggregate injection rate 1 in a channel with n nodes.*

22

Proof: First let us consider the case of execution in which each stage ends. Such executions have infinitely many phases and infinitely many blocks.

The total net increase of the number of packets in queues during up to $3n$ silent rounds of a block is at most $3n + w$, where the number $3n$ is due to the injection rate and w to the burstiness $\delta \leq w$. We argue that the number $6n + w$ of packets is an upper bound on the number of packets queued at all times. The argument is by induction on the phase number, where we show the invariant that $6n + w$ is an upper bound on the number of packets in all the queues.

The silent rounds occurring in a makeup stage are accounted for in the next makeup stage, so we may consider only rounds with successful transmissions in a makeup stage for the sake of accounting purposes. These rounds result in suppressing the number of packets in the red nodes. While the number of packets of the red nodes is shrinking over a makeup stage, with possible variations due to burstiness, the number of packets accumulated in the green nodes could be increasing. This has the effect of trading (a decrease of the number of) packets in red nodes for some (increase of the corresponding number of) packets in green nodes. This means that the increase of the number of packets in queues, due to silent rounds, by $3n + w$ packets, may affect both red and green nodes.

For a specific block of up to $3n$ silent rounds, this increase of the number of packets is neutralized in a node when its queue becomes empty. As the neutralization process is spread over at most two consecutive phases, by Lemma 5, the increases contributed by two consecutive blocks may exist simultaneously, which may result in the effect of up to doubling the increase of $3n$ packets, contributed by one block, due to injection rate, while the contribution of w to the bound, due to the burstiness, is a global one-time effect.

Next consider the case of executions in which some stage does not end. It is not the first preparation stage, unless there are no packets to transmit. This stage that does not end would end if only sufficiently many silent rounds occurred. It follows that, in this case, a packet is heard in every round starting from some round t. Any increase of the number of packets starting from round t can be due only to burstiness. Clearly, there exists another execution, that proceeds in exactly the same manner as the considered one up to round t, and which afterwards has the property that each stage ends. The number of queued packets up to round t is the same in both executions. This number of packets is $\mathcal{O}(n + w)$, because one of these executions has infinitely many phases and the bound $\mathcal{O}(n + w)$ on the number of packets in queues applies to any round in it. $\square$

A block of silent rounds may result in an increase of the number of packets in queues of red nodes, as compared to the beginning of the immediately preceding pure stage. A makeup stage is there to alleviate this effect.

Lemma 7 *A makeup stage of algorithm* COLORFUL-NODES *takes* $\mathcal{O}(nw)$ *rounds against adversaries of window* w *and aggregate injection rate* 1 *in a channel with* n *nodes.*

Proof: We denote by G and R the sets of nodes that begin a makeup stage as green and red, respectively. Let g equal the sum $\sum_{i \in G} C[i] = g$ of the entries of the array C over the green nodes, and r be the similar sum $\sum_{i \in R} C[i] = r$ of the entries of the array C over the red nodes, as at the start of the makeup stage. We have $g + r = \gamma$, because G and R make a partition of all the nodes. By Lemma 1, the sets G of green nodes and R of red nodes have had packets injected into them with the cumulative rates of at most g/γ and r/γ, respectively, during the previous phase.

The rounds of the makeup stage, when only the red nodes transmit, can be conceptually parti-

tioned into disjoint segments of γ rounds each. We may employ the following accounting to assign roles to rounds in each such a segment. The first r rounds could be considered as devoted to unloading new packets injected into red nodes during this segment. The remaining g rounds could be treated as accounting for unloading packets injected into red nodes during the preceding block of silent rounds.

There are some p packets in red nodes in the beginning of this makeup stage. We need to account for the packets injected into these nodes during at most two phases, as each node gets its queue empty at least once during a phase. A phase contributes at most $3n$ silent rounds, which translates into at most $6n$ packets, by Lemma 5. There could be a surge of up to w packets injected due to burstiness, but this fluctuation needs to be compensated by the corresponding number of rounds that follow within the window of w rounds with no packets injected in them, due to the definition of a window-type adversary. This means that such surges are compensated by the adversary's behavior and may contribute up to w to packet latency. We conclude that we can use $p = 6n$ in our estimates of the duration of a makeup stage, as we account for rounds spent on transmissions compensating rounds when the adversary could inject.

It takes $\frac{p}{g}$ segments of γ rounds each to dispose of all the packets in the red nodes, possibly increased by at most w rounds due to burstiness. This makes $\frac{p\gamma}{g} + w$ rounds of the makeup stage when packets are heard. There are also r silent rounds in this stage. The total number of rounds is therefore $\mathcal{O}(nw)$, because $p = 6n$ and the inequalities $\gamma \le w$, $r \le n$, and $g \ge 1$ hold. $\qquad\square$

The following theorem summarizes the performance of algorithm COLORFUL-NODES.

Theorem 7 *When algorithm* COLORFUL-NODES *is executed against adversaries of window w and aggregate injection rate 1 in a channel with n nodes, then packet latency is $\mathcal{O}(nw)$, while the number of queued packets is $\mathcal{O}(n + w)$ in any round.*

Proof: The estimate of a bound on the number of queued packets in any round is given by Lemma 6. To consider packet latency, we proceed through two cases determined by whether all stages eventually end or rather some stage lasts forever.

Suppose first that each stage eventually ends. We examine how stages of all kinds contribute to the length of phases and through that to packet latency.

Each of the preparation, update and makeup stages contributes at most their own duration to the length of a phase. In the case of the preparation stage, we mean that is over in at most n rounds since the first packet is injected. An update stage takes $\mathcal{O}(n + w)$ rounds, by its design. A makeup stage takes $\mathcal{O}(nw)$ rounds, by Lemma 7.

Next we consider pure stages. There is no general upper bound on the duration of any such a stage. A pure stage ends after n silent rounds. In the course of a pure stage, the rate of transmitting packets is equal to the rate with which they were injected in the preceding phase, by how the array C is used in scheduling transmissions. This means that a packet gets delayed by at most n plus the maximum number of packets queued while this packet is handled, which is $\mathcal{O}(n + w)$ by Lemma 6.

Next we consider the case when some stage never ends. It needs to be a pure stage, as makeup stages are excluded by Lemma 7. Up to this pure stage, a bound on packet latency derived with the assumption that each stage ends applies. The rate of transmitting packets during the last (unbounded) pure stage is equal to the rate with which packets were injected in the preceding

phase, because a packet is heard in every round, starting from some round in the pure stage. It follows that packet latency during such an unbounded pure stage is of the order of magnitude of the number of queued packets, which is $\mathcal{O}(n + w)$ by Lemma 6.

There are finitely many cases, as considered above, each producing a partial bound on packet delay. Each is either of the form $\mathcal{O}(n + w)$ or $\mathcal{O}(nw)$, which gives $\mathcal{O}(nw)$ as the overall bound. □

Observe that the bound on packet latency in Theorem 7 is tight. This is because algorithm COLORFUL-NODES is conflict free, so packet latency can be $\Omega(nw)$ in some executions when the algorithm is executed against adversaries of sufficiently large windows w in systems of sufficiently large sizes n, by Theorem 4.

6   Conclusion

We introduced a model of adversarial queuing on multiple access channels in which individual injection rates are associated with nodes. The partitioning of the aggregate rate among the nodes constrains the adversary but is unknown to the nodes. We developed a number of algorithms for the aggregate rate 1 that transmit packets with bounded latency. The bounds on queue size and packet latency of our algorithms are not expressed in terms of the distribution of the aggregate injection rate among the nodes as their individual rates, but are given only in terms of the number of nodes n and the burstiness, which equals the window size w for the aggregate rate 1.

The purpose of this work was to compare the communication environments determined by multiple access channels in which adversaries are determined by individual injection rates with channels in which adversaries are constrained only by global injection rates, as studied in [22]. In both environments, no property of adversaries is known to algorithms. A comparison of these adversarial models was to be in terms of the attainable quality of broadcasting, for the maximum throughput of 1. The main discovered difference between the globally-restrained and individually-restrained adversaries is that bounded packet latency by non-adaptive algorithms is achievable in an adversarial model in which individual injection rates are associated with nodes, which is impossible for adversaries that are globally-restrained only.

We developed algorithms for a window-type adversary with packet latency that is close to asymptotically optimal in the following two cases. One is when the algorithms are adaptive and channels are without collision detection. Another is when algorithms are non-adaptive but channels are with collision detection. Packet latency of non-adaptive algorithms for channels without collision detection turned out to be more challenging to restrict. The algorithm we developed achieves $\mathcal{O}(nw)$ bound on packet latency. This algorithm avoids conflicts for access to the channel. As we showed, packet latency has to be $\Omega(nw)$ for such conflict-avoiding algorithms. This means that the developed algorithm is best possible in this class in terms of asymptotic packet latency.

The question if a non-adaptive algorithm can achieve packet latency that is asymptotically smaller than nw, for channels without collision detection and for window-type adversaries of individual injection rates, remains open.

The adversarial model considered in this work is of the window type. It is a natural question to ask how to extend the adversarial model of individual injection rates to the general leaky-bucket case, and how would such an adversarial model affect algorithms' performance. In the case of globally-constrained adversaries with injection rate 1, it was shown in [22] that the two models of

window and leaky-bucket adversaries make a difference even for small size systems.

References

[1] N. M. Abramson. Development of the ALOHANET. *IEEE Transactions on Information Theory*, 31(2):119–123, 1985.

[2] C. Àlvarez, M. J. Blesa, J. Díaz, M. J. Serna, and A. Fernández. Adversarial models for priority-based networks. *Networks*, 45(1):23–35, 2005.

[3] C. Àlvarez, M. J. Blesa, and M. J. Serna. The impact of failure management on the stability of communication networks. In *Proceedings of the 10th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 153–160. IEEE, 2004.

[4] L. Anantharamu and B. S. Chlebus. Broadcasting in ad hoc multiple access channels. *Theoretical Computer Science*, 584:155–176, 2015.

[5] L. Anantharamu, B. S. Chlebus, D. R. Kowalski, and M. A. Rokicki. Deterministic broadcast on multiple access channels. In *Proceedings of the 29th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–5. IEEE, 2010.

[6] L. Anantharamu, B. S. Chlebus, D. R. Kowalski, and M. A. Rokicki. Medium access control for adversarial channels with jamming. In *Proceedings of the 18th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, Lecture Notes in Computer Science, vol. 6796, pages 89–100. Springer, 2011.

[7] L. Anantharamu, B. S. Chlebus, and M. A. Rokicki. Adversarial multiple access channel with individual injection rates. In *Proceedings of the 13th International Conference on Principles of Distributed Systems (OPODIS)*, Lecture Notes in Computer Science, vol. 5923, pages 174–188. Springer, 2009.

[8] L. Anantharamu, B. S. Chlebus, and M. A. Rokicki. Adversarial multiple access channels with individual injection rates. *Theory of Computing Systems*, 61(3):820–850, 2017.

[9] M. Andrews, B. Awerbuch, A. Fernández, F. T. Leighton, Z. Liu, and J. M. Kleinberg. Universal-stability results and performance bounds for greedy contention-resolution protocols. *Journal of the ACM*, 48(1):39–69, 2001.

[10] M. Andrews and L. Zhang. Stability results for networks with input and output blocking. In *Proceedings of the 30th ACM Symposium on the Theory of Computing (STOC)*, pages 369–377. ACM, 1998.

[11] M. Andrews and L. Zhang. Achieving stability in networks of input-queued switches. *IEEE/ACM Transactions on Networking*, 11(5):848–857, 2003.

[12] M. Andrews and L. Zhang. Routing and scheduling in multihop wireless networks with time-varying channels. *ACM Transactions on Algorithms*, 3(3):Article 33, 2007.

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

[14] M. A. Bender, M. Farach-Colton, S. He, B. C. Kuszmaul, and C. E. Leiserson. Adversarial contention resolution for simple channels. In *Proceedings of the 17th ACM Symposium on Parallel Algorithms (SPAA)*, pages 325–332. ACM, 2005.

[15] M. A. Bender, J. T. Fineman, S. Gilbert, and M. Young. How to scale exponential backoff: Constant throughput, polylog access attempts, and robustness. In *Proceedings of the 27th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 636–654. SIAM, 2016.

[16] R. Bhattacharjee, A. Goel, and Z. Lotker. Instability of FIFO at arbitrarily low rates in the adversarial queueing model. *SIAM Journal on Computing*, 34(2):318–332, 2005.

[17] M. Bieńkowski, M. Klonowski, M. Korzeniowski, and D. R. Kowalski. Dynamic sharing of a multiple access channel. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS)*, Leibniz International Proceedings in Informatics, vol. 5, pages 83–94. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2010.

[18] M. J. Blesa, D. Calzada, A. Fernández, L. López, A. L. Martínez, A. Santos, M. J. Serna, and C. Thraves. Adversarial queueing model for continuous network dynamics. *Theory of Computing Systems*, 44(3):304–331, 2009.

[19] A. Borodin, J. M. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queuing theory. *Journal of the ACM*, 48(1):13–38, 2001.

[20] B. S. Chlebus. Randomized communication in radio networks. In P. M. Pardalos, S. Rajasekaran, J. H. Reif, and J. D. P. Rolim, editors, *Handbook of Randomized Computing*, volume I, pages 401–456. Kluwer Academic Publishers, 2001.

[21] B. S. Chlebus, D. R. Kowalski, and A. Lingas. Performing work in broadcast networks. *Distributed Computing*, 18(6):435–451, 2006.

[22] B. S. Chlebus, D. R. Kowalski, and M. A. Rokicki. Maximum throughput of multiple access channels in adversarial environments. *Distributed Computing*, 22(2):93–116, 2009.

[23] B. S. Chlebus, D. R. Kowalski, and M. A. Rokicki. Adversarial queuing on the multiple access channel. *ACM Transactions on Algorithms*, 8(1):5:1–5:31, 2012.

[24] V. Cholvi and J. Echagüe. Stability of FIFO networks under adversarial models: State of the art. *Computer Networks*, 51(15):4460–4474, 2007.

[25] A. E. F. Clementi, A. Monti, and R. Silvestri. Optimal *F*-reliable protocols for the Do-All problem on single-hop wireless networks. In *Proceedings of the 13th International Symposium on Algorithms and Computation (ISAAC)*, Lecture Notes in Computer Science, vol. 2518, pages 320–331. Springer, 2002.

[26] J. Czyżowicz, L. Gąsieniec, D. R. Kowalski, and A. Pelc. Consensus and mutual exclusion in a multiple access channel. *IEEE Transaction on Parallel and Distributed Systems*, 22(7):1092–1104, 2011.

[27] G. De Marco and D. R. Kowalski. Contention resolution in a non-synchronized multiple access channel. In *Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 525–533. IEEE, 2013.

[28] G. De Marco, M. Pellegrini, and G. Sburlati. Faster deterministic wakeup in multiple access channels. *Discrete Applied Mathematics*, 155(8):898–903, 2007.

[29] S. Dolev, S. Gilbert, R. Guerraoui, F. Kuhn, and C. C. Newport. The wireless synchronization problem. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 190–199. ACM, 2009.

[30] S. Dolev, S. Gilbert, R. Guerraoui, and C. C. Newport. Gossiping in a multi-channel radio network. In *Proceedings of the 21th International Symposium on Distributed Computing (DISC)*, Lecture Notes in Computer Science, vol. 4731, pages 208–222. Springer, 2007.

[31] R. G. Gallager. A perspective on multiaccess channels. *IEEE Transactions on Information Theory*, 31(2):124–142, 1985.

[32] L. Gąsieniec, A. Pelc, and D. Peleg. The wakeup problem in synchronous broadcast systems. *SIAM Journal on Discrete Mathematics*, 14(2):207–222, 2001.

[33] S. Gilbert, R. Guerraoui, D. R. Kowalski, and C. C. Newport. Interference-resilient information exchange. In *Proceedings of the 28th IEEE International Conference on Computer Communications (INFOCOM)*, pages 2249–2257. IEEE, 2009.

[34] S. Gilbert, R. Guerraoui, and C. C. Newport. Of malicious motes and suspicious sensors: On the efficiency of malicious interference in wireless networks. *Theoretical Computer Science*, 410(6-7):546–569, 2009.

[35] L. A. Goldberg, M. Jerrum, S. Kannan, and M. Paterson. A bound on the capacity of backoff and acknowledgment-based protocols. *SIAM Journal on Computing*, 33(2):313–331, 2004.

[36] L. A. Goldberg, P. D. MacKenzie, M. Paterson, and A. Srinivasan. Contention resolution with constant expected delay. *Journal of the ACM*, 47(6):1048–1096, 2000.

[37] A. G. Greenberg and S. Winograd. 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):589–596, 1985.

[38] J. Håstad, F. T. Leighton, and B. Rogoff. Analysis of backoff protocols for multiple access channels. *SIAM Journal on Computing*, 25(4):740–774, 1996.

[39] T. Jurdziński and G. Stachowiak. Probabilistic algorithms for the wakeup problem in single-hop radio networks. In *Proceedings of the 13th International Symposium on Algorithms and Computation (ISAAC)*, Lecture Notes in Computer Science, vol. 2518, pages 535–549. Springer, 2002.

[40] J. Komlós and A. G. Greenberg. An asymptotically fast nonadaptive algorithm for conflict resolution in multiple-access channels. *IEEE Transactions on Information Theory*, 31(2):302–306, 1985.

[41] D. R. Kowalski. On selection problem in radio networks. In *Proceedings of the 24th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 158–166. ACM, 2005.

[42] Z. Lotker, B. Patt-Shamir, and A. Rosén. New stability results for adversarial queuing. *SIAM Journal of Computing*, 33(2):286–303, 2004.

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

[44] P. Raghavan and E. Upfal. Stochastic contention resolution with short delays. *SIAM Journal on Computing*, 28(2):709–719, 1998.

[45] A. Rosén. A note on models for non-probabilistic analysis of packet switching networks. *Information Processing Letters*, 84(5):237–240, 2002.

[46] A. Rosén and M. S. Tsirkin. On delivery times in packet networks under adversarial traffic. *Theory of Computing Systems*, 39(6):805–827, 2006.