The Impact of Asynchrony on Stability of MAC

1st Paweł Garncarek
Institute of Computer Science
University of Wrocław
Wrocław, Poland
pgarn@cs.uni.wroc.pl

2nd Dariusz R. Kowalski
School of Computer and Cyber Sciences
Augusta University
Augusta, Georgia, USA
dkowalski@augusta.edu

3rd Shay Kutten
Technion, Haifa, Israel
& SIT Fraunhofer & Athena, Darmstadt, Germany
lastname at technion.ac.il

4th Lauren Murach
School of Computer and Cyber Sciences
Augusta University
Augusta, Georgia, USA
lmurach@augusta.edu

2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS) | 979-8-3503-8605-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDCS60910.2024.00023

***Abstract*—A large volume of work has already studied various aspects of a synchronous multiple access channel (MAC). However, synchronization is costly and far from reality. Very little is known in the case when stations communicating on the channel may observe asynchronous behavior. Unfortunately, in certain strong asynchrony settings it is impossible to ensure even a small positive throughput (deterministically). Hence, in this paper, we study whether a limited amount of synchrony is already enough for obtaining stability and high throughput.**

More specifically, we present a novel model to capture a bounded asynchrony, where the "bounded" aspect is captured by an upper bound R on the length of any asynchronous time slot. We design two distributed deterministic algorithms to schedule transmissions of dynamically arriving packets at asynchronous stations, which guarantee optimal throughput for all but one packet injection rates and bounded queues at any time (this combination is sometimes known as optimal stable throughput). One of these algorithms is collision-free, while the other, instead, avoids control messages. Combining these results with our impossibility results we characterize exactly the very limited case where there is an inherent difference between synchronous and asynchronous networks for obtaining optimal stable throughput for this problem. As a subroutine, we design a new leader election algorithm for this model and prove upper and lower bounds on the number of slots. Interestingly, when R is a constant, our results match (asymptotically) the known results in synchronous slotted networks, while if R is a larger parameter, our lower bound proves that an additional factor of $\Omega(\frac{R}{\log R})$ is necessary in the formula on the number of slots.

***Index Terms*—Multiple access channel, Bounded asynchrony, Leader election, Dynamic packets injection, Packets' transmission problem, Stable throughput.**

I. Introduction

Shared channels have been investigated intensively for a long time (e.g., [1]). They occur not only in radio and LAN networks but also in other contexts that involve broadcast where mutual exclusion would be desirable. Examples range from body cells (that may secrete some molecules that can be detected by other cells [2]) to various variants of PRAM ([3], [4]), multiprocessor computing ([5]), or even blockchains.

Informally speaking (for now, definitions appear later), when a network station transmits alone, its message is delivered to the stations on the channel successfully. However, transmissions (of more than one station) that overlap in time are lost.[1] and the channel capacity is thus essentially wasted.

Most of the research concentrates on synchronous networks. The idea is to partition the time into equal-duration periods (slots). Every slot is assumed to start (and ends) simultaneously for all stations and is long enough for the transmission (and reception) of exactly one packet. Good performance has been shown for such synchronous networks. For example, even though packet transmissions are sometimes unsuccessful (so the channel is sometimes effectively wasted), it was shown e.g. in [6]–[10] that, even with deterministic protocols, synchronous slotted networks guarantee a max stable injection rate of 1, that is, the throughput there is the same as the average arrival rate, even for a packet arrival rate as high as 1 packet per slot.[2] Moreover, this is also a *stable output*, that is [6], output obtained with the length of the queues of packets to be transmitted at the stations being bounded, see e.g. [6], [11].

Much less research has been performed regarding networks that are less synchronous. This is likely to have happened because of the much poorer performance obtained. Some asynchrony did appear already in the famous pioneering Aloha project [12]; specifically, synchronized slots were not assumed. Each station chose a random time to transmit. It was shown later that adding more synchrony, namely, using slotted Aloha, obtained stability even for a much higher arrival rate [1].

Unfortunately, adding synchronization incurs various costs, sometimes to the point where tight synchronization is not feasible. For example, tight synchronization may be too costly for weak devices such as those in a sensor network (e.g., [13]). It may also be too costly when the shared channel models access to some remote, or a distributed resource, or

The second author was partially supported by the by NSF grant CNS-2131538. The third author is supported in part by the Israeli Science Foundation and his research was carried out in part within the framework of the Grand Technion Energy Program (GTEP)

[1]There exist many model variations regarding channel feedback.

[2]Note that the throughput cannot exceed 1 because at most one packet can be successfully transmitted in a slot.

2575-8411/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDCS60910.2024.00023

the case where the propagation delays vary significantly (e.g. because of significant differences of the distances of stations from the shared resource). In other cases, synchronization has evolved in nature but has not become very tight (e.g., [14]–[16]). Attackers may pose another hurdle to synchronization. Differences in speed may also be harder to avoid when the channel access is performed by software, rather than by specialized hardware (such as in SDN, e.g., [17]). Synchronization also involves costs such as using (costlier) dedicated hardware (rather than using asynchronous processes controlled by an operating system), losing some of the bandwidth on control fields and signals, causing faster devices to wait, etc. To complete the picture, note that even under tight synchronization, the synchronization is never perfect. This paper asks whether high stable throughput can be obtained with a much lower level of synchrony.

A. Contributions

We formulate a partially asynchronous slotted channel model. We still assume that each station i can transmit exactly one of its packets in each of i's slots. (More formal definitions appear later in Section II.) However, the actual slot lengths are controlled by an online adversary who can make the decision about when to end a slot. A slot's length in an execution can vary from 1 to some r that is unknown to the algorithm but is upper bounded by some $R \geq 1$ that is known.[3] We do not assume that stations can measure the lengths of the slots.

We assume channel sensing (a weaker form of collision detection) and the ability to use acknowledgments – these are very popular assumptions in the design of shared channel communication protocols, motivated by classes of TDMA and CSMA/CA collision-avoiding wireless protocols or ICMP and WiFi control frames, see e.g., [1], [18]. Additionally, we consider the impact of two other popular model features: the ability to send control messages and the requirement for collision avoidance.

As a technical contribution, we present deterministic algorithms, lower bounds, and instability results for packet transmission – see Table 1 for a summary. Our algorithms are presented in a form of diagrams, for two reasons – to focus better on responding to channel events and adversarial packet injections, and also since it helps to code the algorithms in most of programming languages. Alternative forms, the pseudocodes, are presented in the full version of this paper [19].

Single successful transmission (SST): The first problem we study (SST), is really a form of *leader election*. Multiple stations may have messages to transmit, and the protocol makes sure that exactly one of them succeeds. The analogous problem has been studied for synchronous channels in several settings (e.g., [20]–[22]). ABS, our protocol for SST, is presented in Section III. The maximum number of slots (possibly of different lengths) experienced during the protocol by any station is $O(R^2 \log n)$, see Theorem 1. We show that this is optimal up to an $O(R \log R)$ factor, see the lower bound $\Omega\left(R \cdot \left(\frac{\log n}{\log R} + 1\right)\right)$ in Theorem 2. Interestingly, for a constant R, these bounds match (asymptotically) the known results for protocols relying on synchronized slots [20], [23]. On the other hand, our lower bound shows that the impact of bounded asynchrony with upper bound R to a simple but fundamental leader election problem is with a factor at least $\Omega(\frac{R}{\log R})$.

Intuitively, what we would have like to do in the leader election protocol, is to have a station transmit if, say it has 1 in its least significant bit. Then, a station with a zero there would hear this transmission and drop out of the competition. However, asynchrony can make the stations' relative speeds different and unknown which makes it hard to know when to listen. Moreover, the uncertainty regarding "where in its execution is a station" accumulates as long as the station is silent. The number of different combinations of slot sizes in a schedule grows exponentially with the length of the silent period of a schedule. The simple trick is that every station must transmit rather often during the protocol as long as it competes for leadership. This allows the other stations to estimate the ratio between the slot times. Hence, the algorithm manages to re-balance the search between groups of stations even though those use different waiting times.

The proof of the lower bound, on the other hand, uses a novel technique of "mirror executions" created by the adversary under bounded asynchrony, to prevent stations from transmitting alone (and thus winning).

While the ideas behind algorithms and lower bounds may look simple at first glance, the technical details are often challenging due the fact that the number of intersections among transmitting slots increases by a factor exponential in R. In particular, it is not immediately clear how to solve the SST problem optimally with respect to both R and $\log n$ (in the synchronous case, $R = 1$ and all it takes is a simple binary search using collisions, in $\Theta(\log n)$ rounds, see [20], [23]).

Packets transmission problem (PT) under dynamic packets' arrivals: The main set of results considers the PT task where (a possibly infinite number of) packets are injected ("arrive") to the system dynamically, each to the queue of one station, at times that are selected by an adversary. The protocol is required to transmit all the packets. The (rather common) measure of success used is *Max Stable Rate* (MSR): the maximum injection rate ρ of packet injection under which the protocol still obtains *stability* (that is, there exists an upper bound on the number of packets that were already injected but have not yet been transmitted successfully).

A part of our contribution is the adaptation of the definition of the arrival rate to the current case where packet transmission durations are not equal, (see Section II). As a motivation, consider the following example: if the transmission of a certain packet by station v requires, say, two of the slots of station u then clearly, no network can have stable throughput if one new packet is injected at v in every slot. An injection rate of 1 should in such case be defined to have a new packet injected (on average) every *two* time slots of u. This would keep u fully busy at all times. Let us emphasis that such weights are

[3] R may result, for example, from timeouts used in practice.

Allows Ctrl. Msg.	Allows Collis.	Our Contribution for $R > 1$	Stable ρ	Queue Sizes	Sec. / Thm.	State-of-Art $R = 1$	Stable ρ	Queue Sizes	Ref.
No	No	*Instability*	no $\rho > 0$	N/A	V / 4	RRW	any $\rho < 1$	$\frac{\rho n+b}{1-\rho}$	[11]
No	Yes	AO-ARRoW	any $\rho < 1$	$\frac{2rR^4 n \log n+b}{(1-\rho)^2}$	IV / 3	MBTF	any $\rho \leq 1$	$2(n^2+b)$	[6]
Yes	No	CA-ARRoW	any $\rho < 1$	$\frac{2nR^2+b}{1-\rho}$	VI / 6	MBTF	any $\rho \leq 1$	$2(n^2+b)$	[6]
Yes	Yes	*Instability*	no $\rho = 1$	N/A	V / 5	MBTF	any $\rho \leq 1$	$2(n^2+b)$	[6]

Fig. 1. A summary of main technical results. The first two columns state main model features – algorithms allowing control messages and/or collisions could be more efficient than those without these features. The next four columns describe our results for bounded asynchrony ($R > 1$) – name, range of max stable rate (MSR) ρ, upper bounds on queue sizes, and the references to the section and theorem. Our results are stated in four rows, starting from the most restrictive model features (first result, No control messages, and No collisions allowed) to the most efficient ones (last result). Performance formulas are in simplified forms and the formula for AO-ARRoW assumes we use ABS algorithm for leader election. The last four columns describe the known results in the synchronous setting ($R = 1$), for corresponding model features (for comparison) – the ranges of max stable rate (columns 4 and 8) are, surprisingly, different for bounded asynchrony ($R > 1$) and for full synchrony ($R = 1$), which shows a fundamental difference between synchronous and (bounded) asynchronous channels.

not known to the algorithm, which still needs to adapt and control queue lengths.

In Sections IV and VI, we design two distributed algorithms to schedule transmissions of ongoing injection of packets that guarantee max stable rate close to the optimal 1. One of these algorithms, AO-ARRoW, is not allowed to send control messages, it sends only genuine packets; on the other hand, collisions may occur, and the algorithm mitigates the impact of collisions online. The other algorithm, CA-ARRoW, avoids collisions completely (i.e., no two transmitting slots overlap in any execution of the algorithm); however, it uses additional control messages.

In Section V, we prove complementary impossibility results (1) if a protocol obeys both constraints (that is, does not allow control messages nor collisions), then it cannot guarantee a positive max stable rate, and (2) that even with both control messages and allowing collisions, no protocol can achieve max stable rate exactly 1. These are crucial differences from the synchronous channel see [6], [7], [11]. There, max stable injection 1 can be obtained, and this, even without control messages and with collision avoidance.

We comment that in the table, we do not compare directly our results to those obtained for traditional protocols such as Aloha, slotted aloha or CSMA because of the significant differences between the models. Still, it may be interesting to note that for the task of ongoing transmission of multiple packets by multiple stations, we obtain stability for rates as high as desired, while if one uses Aloha for that task, stability, as mentioned above, was obtained only for a rather low rate [1], [12]. Moreover, combining our algorithms and our impossibility results, we characterized the difference between synchronous and asynchronous networks: both can achieve the same throughput while keeping bounded queues (and for a constant R, also the same asymptotic queue sizes) except for the case where injection rate is exactly 1. (However, for the subroutine task of leader election, that is, of transmitting a single packet successfully, the model of Aloha allows a shorter time).

B. Related Work

Various papers addressed the performance (and other measures of stability) of networks that are asynchronous in the sense that the time is not split into slots, see e.g., [24]. Other papers addressed the impact of coding, e.g., [25], a subject not addressed here. Cover et al. [26] considered random codes on an asynchronous channel with two players, showing that the error converges to zero. Another example of a two-user asynchronous channel coding includes work by Yemini et al. [27]. Generalization to more than two players could be found in Zhang et al. [28] and Shahi et al. [29].

Chlebus and Rokicki [30] addressed a different problem of transmitting a small number of packets across a multi-hop topology, (not every station could hear every other station), in a different definition of asynchrony. Still, they proved various lower bounds showing various inefficiencies resulting from asynchrony.

Many papers considered a channel with the same slots' length but shifted arbitrarily – different wake-up times with a single packet each, no global clock, randomized protocols, latency, and throughput. For references, cf. De Marco and Stachowiak [31].

II. Additional Definitions

We consider a Multiple Access Channel (MAC) with n stations, each having a unique integer ID in set $[n] = \{1, \ldots, n\}$.[4]

Stations attempt to transmit their packets successfully on the channel. For that, we model the channel as a "base station" that receives the packet upon a successful transmission but does not participate in the protocol otherwise. The channel (successfully) receives a transmission T made by a station only if no other transmission occurs at a time that overlaps with T (cf. [1]). Note that, for simplicity, we assume that the time at base station is continuous.[5]

Each station is associated with a partition of the time axis $\mathbb{R}_{\geq 0}$ into (finite) intervals, referred to as *slots*, so that for each

[4]Our results also hold if IDs are in $[n^c]$, for some constant $c > 1$.

[5]Otherwise, one would need to define whether two transmissions in the same slot of the base station collide even if their times do not overlap; this can be solved but is less simple to describe.

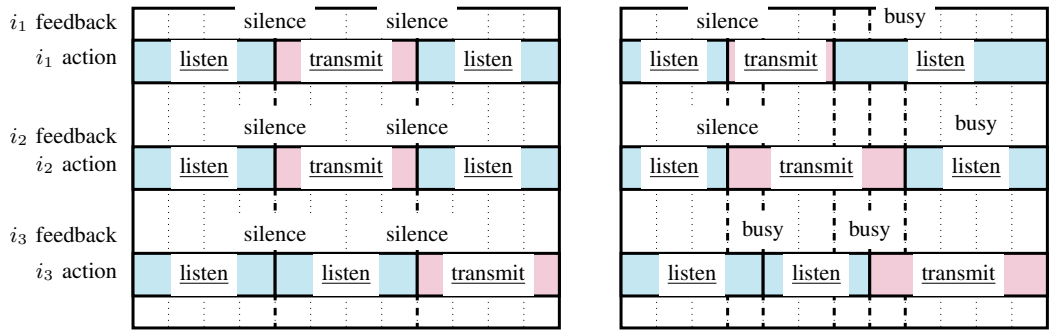

Fig. 2. The figure shows a possible transmission schedule of three stations i_1, i_2, i_3 – in synchronous execution (on the left) and in asynchronous one (on the right). Between the slots of each station, the station receives feedback based on the actions of all the other stations. A dotted line represents checking the actions of all other stations. This feedback (provided to a station at the end of each of its slots) is either silence or busy (for a listening station) or ack (for a station that just finished transmitting successfully) and is shown above the corresponding station's action. The synchronous execution successfully solves SST (station i_3 transmits successfully in its third slot) while the asynchronous execution does not solve SST (potentially, asynchrony requires more slots).

station $i \in [n]$ and index $j \geq 1$, the j-th slot of i is denoted by $S_i^j = [s_i^j, s_i^{j+1})$. These partitions are determined by the adversarial scheduler so that each station $i \in [n]$ is only aware of its own partition, in the sense that i knows (in real-time) whenever slot S_i^j ends and slot S_i^{j+1} begins, but does not know about slots of other stations.[6] In particular, stations do not possess a global clock (nor even a local one).

A station can be either listening or transmitting. (We also use the status "idle" in the hope that this may make the intuition of some algorithms clearer, however in this paper, "idle" is equivalent to "listening".) The feedback of the channel is such that at the end of a slot, both the transmitter and each listening station i know whether a successful transmission ended in the slot. (This transmission may have started in some previous slot of i.) We term this feedback *acknowledgment*, or *ack* for short. If no transmission ends successfully, then station i can distinguish between the case that the slot was *silent* (no station's transmission overlapped) and the case that there were one or more transmissions there. (But it cannot know whether there was an interference, or if so, whether the interference lasted during the whole slot or just during a part of it, etc.). The latter feedback is called *busy (channel)*. See Fig. 2 for an illustrative example.[7]

For a given adversarial schedule of delays, we denote by $r \in [1, R]$ the supremum of slot times over all time slots in that schedule.Stations know bound R but not the value of r.

Definition 1 (Leaky-bucket adversary with cost): A leaky-bucket adversary is described with two parameters: injection rate ρ and burstiness b. Injection rate ρ denotes how often packets may arrive on average. Burstiness b denotes how many packets can be injected in a short period of time.

[6]This means, for example, that slots of different stations can partially overlap and that the j-th slot of one station may have no overlap with the j-th slot of another station.

[7]Note that our definition of channel feedback is consistent with classic communication models, e.g., IEEE 802.11, and is slightly weaker than the notion of Collision Detection used in the theory of radio networks [20] (which allows each station to detect simultaneous transmissions of many stations).

The *cost* of a packet p is the amount of time that the slot used to successfully transmit p eventually take.[8] During any window of t time, all stations together may receive packets of up to $\rho \cdot t + b$ cost.

III. Single Successful Transmission (SST)

An algorithm for solving SST efficiently is now presented, as well as general lower bound on the number of slots of any deterministic solution in our setting. This algorithm is used as a subroutine in our PST protocols in Section VI.

A. Description of algorithm ABS (Asymmetric Binary Search)

ABS is parameterized by the upper bound R on the length of a slot. It implements a distributed binary search (for the minimum ID) on a partially asynchronous channel. In more details, the actions of a station are divided into phases. In the ith phase, the station compares the ith bit of its ID (starting from the least significant one) to that of others by possibly transmitting and by listening to the channel feedback (before and after the transmission or instead). If the station's bit b at the ith position is 0 then the station listens for a relatively short time (for $\texttt{threshold} = 3R$ slots or until sensing busy channel) while in the case that the bit value is 1 – it may take up to $\texttt{threshold} = 4R^2 + 3R$ slots. In the case that the channel was all silent during the listening period, the station attempts to transmit itself – if the transmission is successful, i.e., the station gets an acknowledgment (ack) feedback from the channel at the end of the transmitting slot, then it exits the algorithm "with winning"; otherwise, it repeats the above procedure with the next bit from its ID (i.e., it executes the next phase). A station that at any point hears a collision while not transmitting itself, drops out of the algorithm (exits "by elimination"). See Figure 3 for a simplified diagram.

Notation. Underlined commands, transmit and listen, denote corresponding operations on the channel; each such operation starts at the beginning and finishes at the end of a slot.

[8]This time is not known to the stations and does not have to be decided until the actual transmission.

Recall that all local operations (not underlined) are done in-between of two consecutive slots. Italic commands are typically used for channel feedback, received and processed by a station at the end of the slot. Typewriter font text is used for local variables. The values of these variables are unique to each station and cannot be accessed by other stations unless transmitted over the channel.

a) Analysis of algorithm ABS: A station with ID i that has not exited the algorithm is called *alive* or *active*. First, observe that each phase at any (alive) station i takes at least two slots: at least one in the loop waiting for silent feedback from the channel, see box (1) in Figure 3, and at least one in the loop listening for `threshold` slots (or until busy channel), see boxes (3) and (4). Moreover, if the station does not exit the algorithm at the end of the phase (i.e., it switches to the next phase), the phase has at least $3R + 2$ slots, because it means that the second listening loop lasted exactly `threshold` slots (and `threshold` $\geq 3R$) and there is an additional one transmission slot at the end of the phase.

Next, we prove some auxiliary lemmas. The following lemma captures the main point of the algorithm. On the face of it, since the slot sizes of different stations differ by unknown amounts, they could have reached their corresponding phase h at different time, so they would not have been able to compete according to the value of the hth bit of their respective IDs. The following lemma shows that the algorithm does manage to synchronize them. The rest of the proof of the algorithm is more standard. Some of the following proofs are just sketches. More detailed proofs appear anonymously in [19].

Lemma 1: All alive stations start their corresponding phases within r time of each other.

Proof sketch: Proof by induction on the number of phases. In the first phase, all the stations wake up simultaneously, so the lemma holds. Next we assume that all the alive stations entered some phase h within r time of each other and we show that all the stations that remain alive after h enter the next phase $h + 1$ within r time of each other.

Consider the first station s to start a transmission during the current phase h (see box (5)). Let us call that time t. Since the other stations entered phase h at most r time later than station s did, they are all in the listening loop (box (3) or (4)) by time t. All the stations that did not start their transmission by time t will hear that the channel is busy during their listening loop (box (3) or (4)) and therefore exit the algorithm. Only stations that started transmitting at time t can remain alive. Which means that all the stations that enter the next phase end their transmissions (and thus begin the next phase) within r time of each other. ■

Next, consider a liveness property.

Lemma 2: For any time $t > 0$, if there was no successful transmission in the execution of algorithm ABS by time t, i.e., no station has exited with winning, then there is at least one station executing the algorithm at time t (i.e., the station that has not exited by elimination).

Proof sketch: Assume the converse. Since there was no successful transmission, some station s must have exited by elimination (box (6)). That means that s heard that the channel was busy (in box (3) or (4)), i.e., some other station s' was transmitting (box (5)). Station s' either exits by winning or survives to the next phase. A contradiction in both cases. ■

The next two lemmas are used to show progress.

Lemma 3: Consider a phase h at live stations. Denote by B_0, B_1, a set of alive stations that have their bit b equal 0 or 1, respectively, in their phase h (in box (2)). If both sets are non-empty, then all stations in B_1 will no longer be alive at the end of their phase h.

Proof sketch: Note that at least one station in B_0 transmits before any station in B_1 does. Hence, the latter hear the channel is busy and exit the algorithm. ■

Lemma 4: No two stations transmit in disjoint time slots in any phase.

Proof sketch: Assume by contradiction that some station s_1 ends its transmission at time t_1 before some other station s_2 starts its transmission at time $t_2 > t_1$. Hence, station s_2 hears busy channel while s_1 transmits and s_2 exits by elimination. It means that s_2 does not reach time t_2 and cannot start a transmission at t_2, which contradicts our initial assumption. ■

Lemma 5: Each phase of a station takes $O(R^2)$ slots.

Proof: Consider a phase h and station i which is alive in it. By Lemma 1, all stations start their phase h at times differing by at most r.

Let us analyze the first listening part, i.e., the part corresponding to box (1). Station i first listens (box (1)) until hearing a silent channel. It can hear a busy channel only if a transmission of some other station i' in its previous phase $h-1$ overlaps the listening slot of station i – this is because a single phase takes at least $3R + 1$ listening slots (at least $3R$ from box (3) or (4), plus at least 1 from box (1)) before a single transmission, thus i' could not be in a transmitting slot of any phase smaller or larger than $h - 1$. Thus, the transmission of station i' must have started before station i entered phase h. That transmission will end at most r time after i entered phase h. Listening for r time may take at most r slots. Then, station i will listen 1 more slot in order to finally hear silence. This means that the listening part (box (1)) of station i takes at most $r + 1 \leq R + 1$ slots.

The next part takes at most $4R^2 + 3R$ time slots, by definition. Finally, there is at most one more transmitting slot. Altogether, the total number of slots is $O(R^2)$. ■

Theorem 1: Algorithm ABS solves the SST problem in $O(R^2 \log n)$ slots, corresponding to $O(R^2 r \log n)$ time.

Proof: The number of time slots and the corresponding time bound come directly from the description of the algorithm and by Lemma 5 – there are $O(\log n)$ phases, each having $O(R^2)$ slots. Let us now prove correctness.

As the first observation, we show that if there is a successful transmission of some station i in its phase h, then all alive stations exit with winning or by elimination by the end of their phase h. Suppose, to the contrary, that at least one alive station, say j, does not exit. It means that it attempted to transmit at the end of its phase h but has not succeeded. This

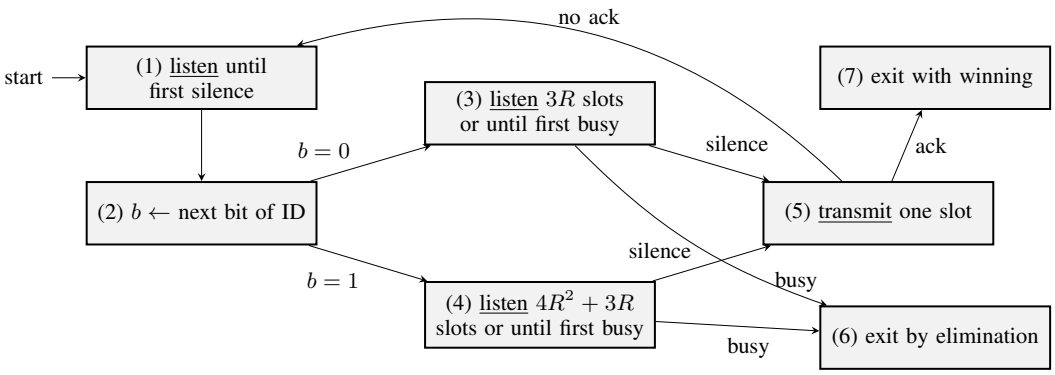

Fig. 3. Diagram of an automaton defined by algorithm ABS(R) at station with binary ID. Labels (1), ... in the beginning of each box are for reference purpose.

transmission cannot overlap with the transmission of station i, as otherwise, station i would not be successful. Consequently, we have at least two non-overlapping transmissions in phase h, contradicting Lemma 4.

Second observation, for any fixed phase h, there is at most one station exiting its phase h with winning. Suppose, to the contrary, that there are at least two such stations, call them i, j, in some phase h. It means that both their transmissions succeeded, so they cannot overlap, which contradicts Lemma 4; thus the second observation is proved.

By straightforward inductive argument, we deduce from the first and the second observations that there could be at most one station exiting the ABS algorithm with winning.

It remains to prove that there is at least one station exiting the protocol by winning. Indeed, by the second observation, it will be a unique station in the whole protocol, and by the first observation – when it exits in some phase h, all other stations also exit (but by elimination) by the end of their phase h. Suppose, to the contrary, that no station exits ABS by winning. By Lemma 2 applied to the very last time t, there is at least one station that does not exit the protocol by elimination. So it remains to show that some station s is eliminated, to show progress. Consider the sets of stations that are alive at the beginning of their last phase of ABS, and partition them into sets B_0 and B_1 depending on the value of their bit b in that phase being 0 or 1, respectively. Only the following three cases could occur:

Case 1: $|B_0| > 1$ or $|B_1| > 1$. The argument is symmetric, thus w.l.o.g. we may assume $|B_0| > 1$. Consider some $i, j \in B_0$. They both have 0 as the value of the currently processed bit of their ID, therefore their IDs must differ in some earlier position h. W.l.o.g. assume that bit h of station i is 0 while bit h of station j is 1. Consider their phases h. They were both alive at that phase, and thus by Lemma 3, station j would no longer be alive at the end of its phase h. This is a contradiction with the assumption that j is alive at the beginning of the very last phase – thus this case cannot hold.

Case 2: $|B_0| = |B_1| = 1$. Directly from Lemma 3, the station in B_1 would stop being alive at the end of phase h, making this case contradictory.

Case 3: $|B_0| + |B_1| = 1$. If only one station, say i, is alive in the last phase, then no other station intervenes and it successfully transmits and exits with winning, which is again a contradiction finalizing the proof that at least one successful transmission occurs. More precisely, let t be the time when station i starts the last phase. By Lemma 1, all other stations have exited by time $t+r$. As all of them exited by elimination, there was no successful transmission in the preceding phase. Hence, throughout the whole last phase the channel is silent, unless station i would transmit. Such transmission happens, as station i does not sense a busy channel during its listening period, and as no other station transmits during the last phase of station i, this transmission is successful. This completes the proof.

■

B. Lower Bound for the SST Problem

Our key concept in proving a lower bound is based on mirror executions, in which every participating station receives feedback that "mirrors" its channel activity. More precisely, an execution of an algorithm on a (partially asynchronous) channel is a *mirror execution* if for any slot of any participating station, the channel feedback is silence if the station listens, and "busy channel" without an acknowledgment otherwise. Observe that as long as the execution is a mirror one, there is no successful transmission of any station taking part in it.

Theorem 2: Any deterministic algorithm solving the SST problem requires $\Omega\left(r \cdot \left(\frac{\log n}{\log r} + 1\right)\right)$ slots. In particular, if the adversary chooses $r = R$, the lower bound becomes $\Omega\left(R \cdot \left(\frac{\log n}{\log R} + 1\right)\right)$, and for a constant R it is $\Omega(\log n)$ – the same as on a purely synchronous channel.

Proof: We need to show that there is an execution of some stations in which one of the stations does not terminate before finishing its $r \cdot (\frac{\log n}{\log r} + 1)$ slots. We may restrict the discussion to the case of $R \geq r \geq 4$, as otherwise both R, r would be constant and we could apply a logarithmic lower bound $\Omega(\log n)$ that holds for some worst-case execution even if slots are synchronized, cf. [23].

Until the last paragraph of this proof, we assume $n \geq 2r$; the remaining case will be analyzed at the very end. The adversary constructs an execution in an online manner, in consecutive synchronized phases, each consisting of r slots of each participating (i.e., alive) station. At the beginning of each phase h, at time t_h, there is a set of alive stations C_h. We assume $t_1 = 0$ and $C_1 = [n]$. The adversary aims to preserve the following invariant at the beginning of phase $h + 1$:

- $C_{h+1} \subseteq C_h$ and $|C_{h+1}| \geq n/(2r)^h$;
- $t_{h+1} = t_h + r \cdot g$, for some $g \in [1, r]$;
- there is a mirror execution in which all stations in C_{h+1} participate, each having r slots in each phase, and which finishes by time t_{h+1} (and hence, no successful transmission of any of them).

The proof of the invariant is by induction on the phase number h. At the beginning of phase 1, it clearly holds as $C_1 = [n]$.

Suppose the invariant holds up to the beginning of phase h – we prove that it also holds at the beginning of phase $h+1$. For all the stations in C_h, the adversary computes their transmit/listen schedule in the mirror execution guaranteed by the invariant by time t_h. Then, it extends each of them by subsequent r slots, under the assumption that in each slot the channel feedback is mirroring their transmit/listen choice. This extension is meanwhile virtual, in the sense that so far, we have not shown that it may exist in a real execution – however, we will show that for some of the stations in C_h it is possible to obtain such an extension of their actions and together they result in a mirror execution.

Specifically, let ζ_i be such an extension of activities of station $i \in C_h$ for phase $h + 1$ – again, it is defined by the algorithm run at station i, under the history of previous phases and the virtual assumption that channel feedback mirrors the station action. We denote $\zeta_i[x] = 0$ if station i listens in the slot x of the extension, and $\zeta_i[x] = 1$ if station i transmits. Note that we may ignore idle slots as they do not get any feedback and thus could not change the action of the station in the next slot (formally, denote idle slots by $\zeta_i[x] = *$ and in the following analysis w.l.o.g. we could treat $*$ as 0). Let $f(i)$ be the number of interleaved maximal blocks of 0's and 1's in ζ_i if the first maximal block is of 0's, and $f(i)$ be the number of such blocks plus r if the first maximal block in ζ_i is a block of 1's. Note that $f(i) \in \{1, \ldots, 2r\}$, and more precisely, $f(i) \in \{1, \ldots, r\}$ when the first maximal block in ζ_i is a block of 0's and $f(i) \in \{r + 1, \ldots, 2r\}$ if the first block of ζ_i is a block of 1's.

Since there are at most $2r$ different values of $f(i)$, by a pigeonhole principle there is a value g of $f(i)$ such that are at least $\lfloor |C_h|/(2r) \rfloor$ stations i in C_h with value g of function $f(i)$; we put all these stations in the newly created set C_{h+1}. It clearly satisfies the first point of the invariant at the beginning of phase $h + 1$.

If $g \leq r$, the adversary takes any station $i \in C_{h+1}$ and constructs a delay schedule block by block. Specifically, all slots in a given block in ζ_i are uniformly enlarged such that their total length is r. This way, the total time length of the r slots in ζ_i (partitioned into g blocks) is rg. Moreover, blocks are aligned in time across the participating stations in C_{h+1}. Since all ζ_i, for $i \in C_{h+1}$, start with a block of 0's and have the same number of subsequent blocks, a straightforward inductive argument (over the number of blocks in ζ_i) proves that actually blocks of 1's are aligned (in their starting times) with corresponding blocks of 1's at other participating stations in C_{h+1} (and the same holds for corresponding blocks of 0's). Hence, in this execution of stations in C_{h+1}, the blocks of listening slots are aligned and result in feedback "silent", and correspondingly, the blocks of transmitting slots are aligned resulting in feedback "busy without acknowledgment". It yields mirror execution, extended to phase h. Hence, the third invariant is proved, while the second follows from the alignment of g blocks, containing r slots in total (per station), over time rg.

In the case of $g > r$, we use the same argument as above but with respect to $g - r$ blocks instead of g; This is because $g > r$ means that, in the definition of $f(i) = g$, an artificial $+r$ has been applied to the actual number of maximal blocks, in order to distinguish extensions ζ_i that start with a block of 1's from the extensions starting with a block of 0's (the former get values in $\{r + 1, \ldots, 2r\}$ while the latter in $\{1, \ldots, r\}$). This completes the proof of the invariant, as long as the newly defined set C_{h+1} is non-empty. I.e., in at least $\log_{2r} n = \frac{\log n}{\log(2r)}$ phases. Here, in order to assure that the formula on the number of phases is at least 1, we use the initial assumption of $n \geq 2r$.

In case $2r > n$, there is only one phase, in which a single transmitting slot of one station makes the channel busy for any r slots of another station. Hence, the theorem is proved. ■

IV. Packets Transmission without Control Messages, under Dynamic Packet Arrival

We now turn to describe the first algorithm for the on-going transmissions problem PT. Algorithm *Adaptive Order Asynchronous Round Robin Withholding*, or AO-ARRoW in short, is described in Figure 5. Informally, we use the ABS algorithm in Section III-A, as a leader election subroutine. The selected leader (winner) transmits all the packets in its queue and then waits (see below). The losing stations wait until the transmissions from the winner are finished and then compete again in the next leader election.

In order to guarantee stability, and that no stations are starved from transmitting, a station that is done transmitting is not immediately *eligible* to transmit again. A station (whose queue is not empty) becomes *eligible* if it listens for $n - 1$ leader elections Alternatively, to prevent a deadlock (if some stations never have packets), a station becomes eligible if it listens to "enough" silence slots that guarantee that no stations are currently eligible. ("Enough" is called the `threshold` and is equal to the length of the longest possible period of silence during the leader election, note that the number of consecutive silent slots needs to be multiplied by R.)

The main new trick is again one that reduces the uncertainty. In more details, the algorithm synchronizes all the stations that

become eligible, so that they rejoin the competition more or less at the same time. (This is useful for the leader election subroutine). To synchronize, (a) such a station waits for an additional $R\cdot$ `threshold` slots and then (b) transmits a packet. On hearing such a transmission, every station thus waiting to rejoin starts a new round of leader election.

Let A be the length in slots of subroutine Leader_Election(R), B – an upper bound on the time that any station can spend in a long silence with non-empty queue and $S = \frac{nRA+b+B}{1-\rho}$. Let us now define a value that we show later is a bound on the total cost of (all the) the packets that are in (all the) queues at any given time.

$$\begin{aligned} L &= \max\{L_1, L_2\} \text{ , where} \\ L_1 &= S + \frac{(nRA+S)\rho+b}{1-\rho} \text{ , and} \\ L_2 &= (S\rho + nRA\rho + b + B) + (n+1)RA\rho + R\rho + b \text{ .} \end{aligned}$$

Theorem 3: For any adversary with injection rate $\rho < 1$ and burstiness $b \geq 1$, the total cost of packets in the queues in AO-ARRoW never exceeds L.

Using a common term in adversarial queuing theory (e.g. [32]), the theorem shows that AO-ARRoW is *universally stable*. If the Leader_Election(R) used is ABS(R) then we get $A = \log n\left(2R^2+2R+1\right)$ and $B = r(4R^2+3R)R(R+1)+2 = O(rR^4)$.

For proving the theorem, we need some preparations.

Definition 2: Wasted time is a period of time in which no successful packet transmission (by any station) occurs.

Notice that at each time, algorithm is in one of 2 states:

1) at least one station is in boxes (2), (4) or (5) in Fig. 5, or
2) all the stations are in boxes other than (2), (4) and (5).

Definition 3: Let t_1 be a moment in time when the algorithm exits state 2. Let t_2 be the next moment when the algorithm exits state 2 again. The time interval $[t_1, t_2)$ is called a *phase*.

Definition 4: We divide each phase into *subphases*. A subphase starts at the beginning of the phase or at the end of the previous subphase. A subphase ends after n leader elections and their associated transmissions in box (4) or when the algorithm enters state 2.

See Figure 4 for an illustration of an execution of AO-ARRoW in terms of phases and subphases.

Lemma 6: Consider a subphase such that at the start of the subphase, the total cost of packets in all the queues is X such that $X \leq S = \frac{nRA+b+B}{1-\rho}$.

Then, the total cost of packets in all the queues is at most $S + \frac{(nRA+S)\rho+b}{1-\rho} \leq L$ at all times during the subphase. Furthermore, the total cost of packets in all the queues is at most $S\rho + nRA\rho + b$ at the end of the subphase.

Proof: The subphase lasts at most $T = nRA + X + I$ time, where $I \leq T\rho + b$ is the total cost of packets injected during the subphase. It follows that

$$T \leq nRA + X + T\rho + b$$

$$T(1-\rho) \leq nRA + X + b$$

$$T \leq \frac{nRA + X + b}{1-\rho} \text{ .}$$

Therefore, the total cost of packets in all the queues at any time during the subphase is at most

$$\begin{aligned} X + I &\leq X + T\rho + b \leq X + \frac{nRA+X+b}{1-\rho}\rho + b\frac{1-\rho}{1-\rho} \\ &= X + \frac{(nRA+X)\rho + b}{1-\rho} \text{ .} \end{aligned}$$

Using $X \leq S$, we obtain the result in the lemma.

We will now bound the total cost of (all the) packets (in all the) queues at the end of the subphase.

Let $\alpha \leq X + I$ denote the total cost of packets successfully transmitted during the subphase. Note that all packets in the queues at the start of the subphase are successfully transmitted during the subphase, i.e., $\alpha \geq X$.

Note that the subphase lasts $T \leq nRA + \alpha$ time. The total cost of packets in queues at the end of the subphase is at most

$$\begin{aligned} X + I - \alpha &\leq X + (T\rho + b) - \alpha \\ &\leq X + (nRA + \alpha)\rho + b - \alpha \\ &= X + nRA\rho + b - \alpha(1-\rho) \\ &\leq X + nRA\rho + b - X(1-\rho) \\ &= X\rho + nRA\rho + b \\ &\leq S\rho + nRA\rho + b \text{ .} \end{aligned}$$

■

Lemma 7: Consider a subphase such that at the start of the subphase, the total cost of packets in (all the) queues is X such that $X \geq S = \frac{nRA+b+B}{1-\rho}$. Then, the total cost of packets in queues is at most $X + (n+1)RA\rho + R\rho + b$ at all times during the subphase. Furthermore, the total cost of packets in queues is at most $X - B$ at the end of the subphase.

Proof: Suppose the total cost of packets at the start of the phase is $X \geq S$. All the packets that were available at the start of the subphase are transmitted during the subphase, so the subphase lasts at least X time. Let $T \geq X$ denote the length of the subphase. There is at most $W = n \cdot RA$ time wasted during the subphase. Therefore, at least $T - W$ time is spent on successful transmissions. The total cost of packets injected during the subphase is at most $I = T\cdot\rho + b$. Therefore, the total cost of packets at the end of the subphase is at most

$$\begin{aligned} X + I - (T - W) &\leq X + T\rho + b - (T - nRA) \\ &= X - T(1-\rho) + nRA + b \\ &\leq X - X(1-\rho) + nRA + b \\ &= X\rho + nRA + b. \end{aligned}$$

Since $X \geq S = \frac{nRA+b+B}{1-\rho}$, we get $X(1-\rho) \geq nRA+b+B$ and thus $X\rho + nRA + b \leq X - B$, i.e., the total cost of packets decreased by at least B by the end of the subphase.

We will now calculate an upper bound on the total queue sizes at any time during the subphase. Denote the start of the subphase as time 0. Take any point in time t during the

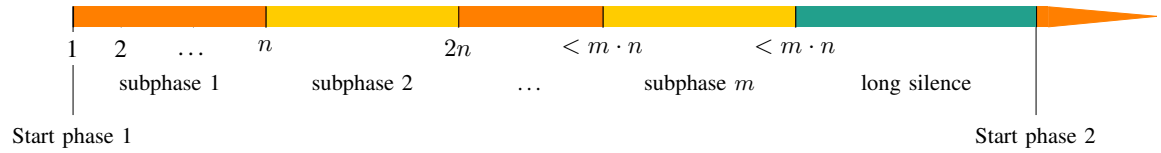

Fig. 4. A timeline representation of how the phases and subphases change as the number of leader elections increases. The numbers on the timeline represent the number of leader elections that passed since the start. The variable n is the number of stations involved in the algorithm, and m represents the number of subphases in the phase (we later prove that m is finite).

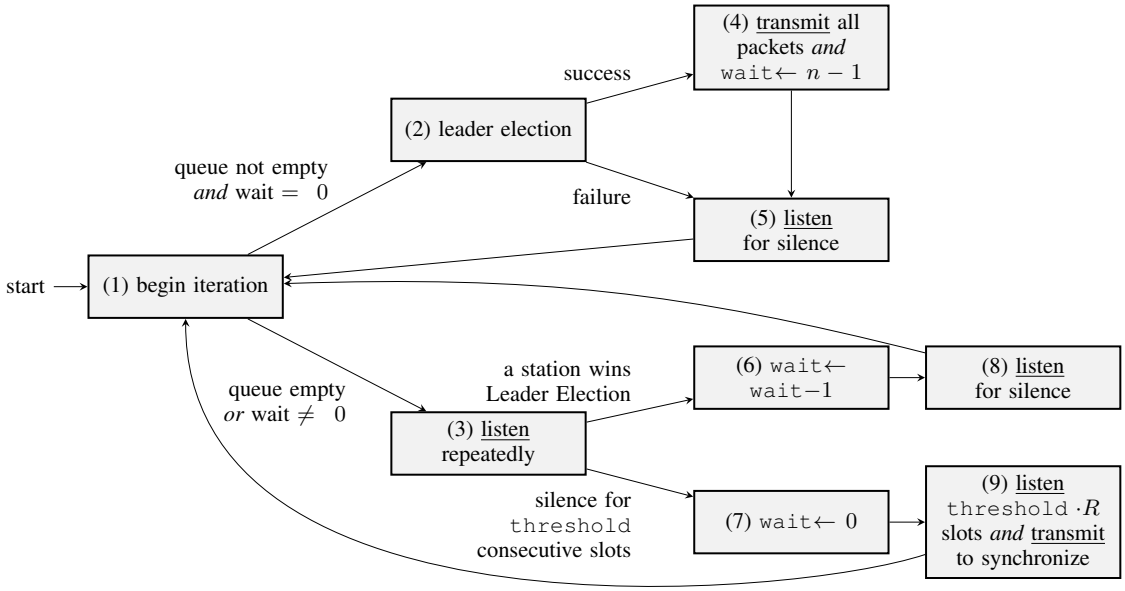

Fig. 5. Diagram of an automaton defined by algorithm AO-ARRoW(R) of a station with a `wait` variable initially set to 0. Numbers in brackets at the start of each box are labels used for reference.

subphase. There were $j \leq n$ leader elections and Y total cost of packets transmitted up to t. Therefore, $t \geq jRA+Y$. In case t is during a packet transmission, we get $t < jRA+Y+R$. In case t is during a leader election, we get $t < (j+1)RA+Y$. In all cases, we get $jRA+Y \leq t < (j+1)RA+Y+R$.

The total cost of packets in queues at time t is at most $C = X + I' - Y$, where $I' \leq t\rho + b$ is the total cost of packets injected during the subphase up to time t. Note that $I' \leq ((j+1)RA+Y+R)\,\rho + b$, so $C = X + I' - Y \leq X + (j+1)RA\rho + R\rho + b - Y(1-\rho)$. We want an upper bound on C. Therefore, we pick the worst case $j = n$ and $Y = 0$. We get that $C \leq X + (n+1)RA\rho + R\rho + b$. ■

Lemma 8: Consider a phase such that the total cost of packets in queues is at most $S\rho + nRA\rho + b + B$ at the start of the phase.

Then, the total cost of packets in queues is at most L at all times during the subphases of the phase. Furthermore, at the end of the last subphase of the phase, the total cost of all packets in the queues is at most $S\rho + nRA\rho + b$.

Proof: We will show by induction that all subphases within the phase start with at most $S\rho + nRA\rho + b + B$ cost of packets in the queues. The base of the induction is trivial since the start of the first subphase coincides with the start of the phase and the total cost of packets at the start of the phase is $S\rho + nRA\rho + b + B$.

Now we present the induction step. Consider any subphase s_1 such that the total cost of packets in queues is at most $S\rho + nRA\rho + b + B$ at the start of the subphase. We will show that the next subphase s_2 (if it exists) also starts with at most $S\rho + nRA\rho + b + B$ cost of packets in the queues.

Let the total cost of packets in queues at the start of subphase s_1 be X. We consider two types of subphases:

- *short subphase* is a subphase such that $X \leq S$;
- *long subphase* is a subphase such that $X > S$.

Consider that s_1 is a short subphase first. According to Lemma 6, the total cost of packets in queues at the end of s_1 is no more than $S\rho + nRA\rho + b$.

Consider that s_1 is a long subphase now. According to Lemma 7, the total cost of packets in the queues at the end of s_1 is no more than $X - B \leq (S\rho + nRA\rho + b + B) - B =$

$S\rho + nRA\rho + b$.

Notice that in both cases, s_1 ends with at most $S\rho+nRA\rho+b$ cost of packets in the queues. Therefore, this is also the cost when the next subphase, s_2, starts.

This completes the induction proof By repeating the argument one more time, we can conclude that at the end of the last subphase of the phase, the total cost of all packets in the queues is at most $S\rho + nRA\rho + b$. Furthermore, according to Lemma 6, the total cost of packets in the queues never exceeds L during any short subphase of the phase. According to Lemma 7, the total cost of packets in the queues never exceeds $(S\rho + nRA\rho + b + B) + (n+1)RA\rho + R\rho + b \leq L$ during any long subphase of the phase. ■

Proof of Theorem 3:

Consider an arbitrary leaky-bucket adversary with costs with injection rate $\rho < 1$ and a burstiness $b \geq 1$. We will show that the cost of all packets waiting in the queues never exceeds L.

We prove by induction that the total cost of packets in queues is at most $S\rho + nRA\rho + b + B$ at each phase's start.

If there is a long silence before the first phase, the long silence lasts at most B time, which means there are at most $B\rho + b \leq S\rho + nRA\rho + b + B$ cost of packets injected before the first phase starts. Otherwise, the first phase starts at the beginning of the execution, and thus there can be at most $0 \cdot \rho + b = b \leq S\rho + nRA\rho + b + B$ cost of packets at the start of the first phase. This completes the base of the induction.

For the inductive step, assume a phase p_1 started with at most $S\rho + nRA\rho + b + B$ cost of packets in the queues. We will show that the next phase p_2 starts with at most $S\rho + nRA\rho + b + B$ cost of packets in the queues as well. Let Z denote the total cost of packets at the end of the last subphase of p_1. According to Lemma 8, $Z \leq S\rho + nRA\rho + b$.

If $Z > 0$, then the long silence lasts at most B time and the total cost of packets in queues at the start of p_2 is at most $Z + B\rho + b \leq S\rho + nRA\rho + b + B$.

If $Z = 0$, then there is at most B time after the first packet is injected during the long silence and before the long silence ends. In this case the total cost of packets in queues at the start of p_2 is at most $B\rho + b$.

In both cases phase p_2 starts with at most $S\rho+nRA\rho+b+B$ cost of packets in the queues. By the power of induction, the total cost of all packets in the queues is at most $S\rho+nRA\rho+b+B$ at the start of every phase.

According to Lemma 8, the total cost of packets in the queues does not exceed L during any phase. Therefore, the cost never exceeds L, which completes the proof. ■

V. Instability Results

We first show the instability of collision-avoidance algorithms without control messages.

Theorem 4: For any injection rate $\rho > 0$ and for any bound $L > 0$ on the maximum number of packets waiting in the system (sum of packets waiting in all stations), for any collision-avoidance algorithm which cannot send control messages, there exists an injection pattern such that the total number of packets waiting in the stations' queues exceeds L.

Proof: Let us fix $L > 0$ and $\rho > 0$. First, the adversary picks two arbitrary stations s_1 and s_2. Only these two stations will receive any packets. The adversary picks a number $S > \frac{2L-1}{\rho(R-1)}$, e.g., $S = \lceil \frac{2L-1}{\rho(R-1)} \rceil + 1$.

The first packet injection into s_1 occurs at the end of slot S of s_1. After that, more packets are being injected into s_1 at the rate $\rho/2$. Let α denote the number of slots of station s_1 that passed since the end of slot number S until the first attempt at packet transmission by s_1, assuming s_1 heard only silence on the channel until slot $S + \alpha$. Note that $\alpha < \frac{2L+1}{\rho}$; otherwise after $S + \alpha$ slots, station s_1 would accumulate over L packets in its queue.

Similarly, the first packet injection into s_2 occurs at the end of slot S of s_2, and more packets are being injected into s_2 at the rate $\rho/2$. Let β denote the number of slots of station s_2 that passed since the end of slot number S until the first attempt at packet transmission by s_2, assuming s_2 heard only silence on the channel until slot $S + \beta$. Again, $\beta < \frac{2L+1}{\rho}$, or otherwise over L packets would accumulate in the queue of s_2 in the alternative scenario where only s_2 receives all packets. Without loss of generality, let us assume that $\alpha \leq \beta$.

Let the adversary fix the length of all listening slots of s_1 to some $1 \leq X \leq R$. Similarly, the adversary fixes the length of all listening slots of s_2 to some $1 \leq Y \leq R$. Note that if

$$(S + \alpha)X = (S + \beta)Y \ , \tag{1}$$

then a collision is generated, i.e., the algorithm is not collision-free. Equation 1 holds if and only if $\frac{X}{Y} = \frac{S+\beta}{S+\alpha}$. Using $0 \leq \alpha \leq \beta < \frac{2L-1}{\rho}$ and $S > \frac{2L-1}{\rho(R-1)}$, we get

$$\frac{X}{Y} \geq \frac{S+\alpha}{S+\alpha} = 1 \geq \frac{1}{R} \ , \text{ and thus:}$$

$$\frac{X}{Y} < \frac{S + \frac{2L-1}{\rho}}{S+0} < \frac{\frac{2L-1}{\rho(R-1)} + \frac{2L-1}{\rho}}{\frac{2L-1}{\rho(R-1)}} = \frac{\frac{1}{R-1} + 1}{R-1} = R \ ,$$

i.e., the adversary can choose $1 \leq X, Y \leq R$ such that a collision (implied by the fact that Eq. 1 holds) is generated.

To summarize, we have noted that if $\alpha \geq \frac{2L+1}{\rho}$ or $\beta \geq \frac{2L+1}{\rho}$, then the corresponding queue will exceed L. Otherwise, i.e., $\alpha, \beta < \frac{2L+1}{\rho}$, the adversary can force a collision, so the considered algorithm is not a collision-avoiding algorithm. ■

Next, consider the impossibility of MSR 1.

Theorem 5: There is no stable algorithm when packet arrivals are controlled by an adversary with arrival rate $\rho = 1$.

Proof sketch: Note that for an algorithm to be stable with arrival rate 1, an algorithm must transmit packets at all times, except possibly, for some finite amount of time. idea behind the proof is to note that whenever one station s_1 stops transmitting to allow another station s_2 to transmit, there may be some wasted time after s_1 stops transmitting and before s_2 starts transmitting. In fact, the adversary can guarantee this will happen by misaligning the slots, using the asynchrony.

Now, the adversary can force the algorithm to change the transmitting station infinitely often just by stopping to inject packets to the transmitting station. Hence, an infinite amount

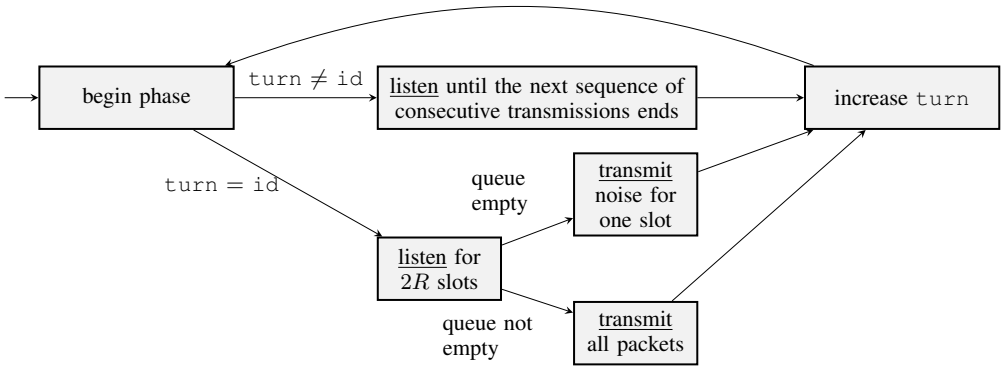

Fig. 6. Diagram of an automaton defined by a single phase of algorithm ARRoW(R). The ID variable is unique to a singular station.

of time is wasted and ALG cannot be stable against adversaries with injection rate $\rho = 1$. ■

VI. Collision-Free Packets Transmission with Control Messages, under Dynamic Packet Arrival

In this section, we consider algorithms that never generate collisions. However, the stations can transmit control messages, i.e., even a station with an empty queue can spend a slot transmitting, which can be utilized as a signal to other stations. We do **not** assume that stations can read the content of transmitted packets, not even that of the control messages (even if the transmission is successful).[9] Note that just the knowledge that a message is being transmitted is useful, even though the message cannot be read.

A. Algorithm description

The algorithm presented in this section is a version of AO-ARRoW protocol – we call it *Collision-Avoidance Asynchronous Round Robin Withholding*, or CA-ARRoW for short. It has two parameters: the space of all IDs, $[n]$, and an anticipated upper bound on asynchrony, R. We allow stations without packets to transmit an empty signal, i.e., give a signal that is identical to packet transmission.

The stations take turns transmitting, cyclically starting from the station with ID 1. Each station has a variable `turn`, which indicates the station ID whose turn it is, and a variable `counter`. A station i, during its turn, transmits all the packets waiting in i's queue or an "empty" signal if i has no packets. Once station $i + 1$ (wrapping cyclically) hears the end of transmissions (hears a silent slot after hearing transmissions) from station i, it waits for $2R$ slots and then begins its turn. A part of station's execution between changing variable `turn` is called a *phase*, see Figure 6 for phase diagram.

Algorithm discussion: Note that, CA-ARRoW is closer in spirit to traditional Round Robin. However, the asynchrony makes Round Robin itself problematic here. Intuitively, whenever a station u hears some transmission, it can upper bound the number of slots that have passed in some other station (using the value of R and the number of slots that passed in u since the previous transmission). It can use this upper bound to avoid collisions. However, if many stations do not have any packets to transmit, the uncertainty accumulates and the upper bound grows exponentially. We use control messages (empty signals) to break long periods of silence.

[9]The motivation comes from various types of encoding, arising from privacy, and various technologies where packets are encoded by some other entities, such as a Base Station, but not by other transmitting stations.

B. Algorithm analysis

Theorem 6: Algorithm CA-ARRoW is universally stable.

The theorem can be formally proven similarly to Theorem 3, but the lack of long silence makes it simpler. The general idea is as follows. For every cycle of n stations taking their turns withholding the channel, there is at most $n \cdot 2R \cdot R$ time wasted ("wasted" according to Definition 2). If a cycle starts with a sufficiently large total cost of packets in the queues $X \geq \frac{2nR^2\rho+b}{1-\rho}$, then more packets will be transmitted than injected during the cycle. Thus, there exists a bound $L = \frac{2nR^2\rho+b}{1-\rho} + 2nR^2 = \frac{2nR^2+b}{1-\rho}$ such that the total cost of packets in the queues never exceeds L.

VII. Discussion and Open Problems

This paper is just an initiation of research into asynchrony and partial synchrony in a shared communication medium. Among the many remaining questions, one may ask is the synchronization we used the minimum one required for achieving high MSR or even throughput? In a sense, this question is similar to the question in the famous work of [33] on the minimum synchronization required for consensus. Can one obtain similar results under assumptions that are relaxed further? (E.g., one may assume that the bound R exists but is not known, and/or collisions cannot be detected by transmitters and/or listening stations and/or if acknowledgments are not provided). Can one do with less knowledge on the system (e.g.,knowing how many stations there are)? Are the methods used here applicable to the case that messages of remote nodes suffer longer delays? One may also check what may the advantages of the use of randomization be (if any).

Further study may also consider cognitive radio, multi-hop networks, or/and the impact of failures. We expect substantially different results than in synchronous channel(s). In addition, we studied and used one simple primitive – leader election (SST) – while it would be interesting to study and apply others (such as point-to-point communication) as well as other network structures (when not all nodes are neighbors of each other). Finally, one should study the practical implications of the proposed algorithms.

REFERENCES

[1] R. Rom and M. Sidi, *Multiple access protocols: performance and analysis*. Springer Science & Business Media, 2012.

[2] Y. Afek, N. Alon, O. Barad, E. Hornstein, N. Barkai, and Z. Bar-Joseph, "A biological solution to a fundamental distributed computing problem," *science*, vol. 331, no. 6014, pp. 183–185, 2011.

[3] B. S. Chlebus, K. Diks, T. Hagerup, and T. Radzik, "New simulations between crcw prams," in *Fundamentals of Computation Theory: International Conference FCT'89 Szeged, Hungary, August 21–25, 1989 Proceedings 7*. Springer, 1989, pp. 95–104.

[4] T. Hagerup and T. Radzik, "Every robust crcw pram can efficiently simulate a priority pram," in *Proceedings of the second annual ACM symposium on Parallel algorithms and architectures*, 1990, pp. 117–124.

[5] M. Herlihy, N. Shavit, V. Luchangco, and M. Spear, *The Art of Multiprocessor Programming*. Morgan Kaufmann, 2020.

[6] B. S. Chlebus, D. R. Kowalski, and M. A. Rokicki, "Maximum throughput of multiple access channels in adversarial environments," *Distributed Comput.*, vol. 22, no. 2, pp. 93–116, 2009. [Online]. Available: https://doi.org/10.1007/s00446-009-0086-4

[7] L. Anantharamu, B. S. Chlebus, D. R. Kowalski, and M. A. Rokicki, "Deterministic broadcast on multiple access channels," in *INFOCOM 2010. 29th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 15-19 March 2010, San Diego, CA, USA*. IEEE, 2010, pp. 146–150. [Online]. Available: https://doi.org/10.1109/INFCOM.2010.5462256

[8] ——, "Medium access control for adversarial channels with jamming," in *Structural Information and Communication Complexity: 18th International Colloquium, SIROCCO 2011, Gdańsk, Poland, June 26-29, 2011. Proceedings 18*. Springer, 2011, pp. 89–100.

[9] M. Bienkowski, T. Jurdzinski, M. Korzeniowski, and D. R. Kowalski, "Distributed online and stochastic queueing on a multiple access channel," *ACM Transactions on Algorithms (TALG)*, vol. 14, no. 2, pp. 1–22, 2018.

[10] L. Anantharamu, B. S. Chlebus, D. R. Kowalski, and M. A. Rokicki, "Packet latency of deterministic broadcasting in adversarial multiple access channels," *Journal of Computer and System Sciences*, vol. 99, pp. 27–52, 2019.

[11] B. S. Chlebus, D. R. Kowalski, and M. A. Rokicki, "Adversarial queuing on the multiple access channel," *ACM Trans. Algorithms*, vol. 8, no. 1, pp. 5:1–5:31, 2012. [Online]. Available: https://doi.org/10.1145/2071379.2071384

[12] N. Abramson, "The aloha system: Another alternative for computer communications," in *Proceedings of the November 17-19, 1970, fall joint computer conference*, 1970, pp. 281–285.

[13] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: a survey," *IEEE network*, vol. 18, no. 4, pp. 45–50, 2004.

[14] A. Jolly, "Breeding synchrony in wild lemur catta," *Social communication among primates*, 1967.

[15] J. Buck and E. Buck, "Synchronous fireflies," *Scientific American*, vol. 234, no. 5, pp. 74–85, 1976.

[16] S. J. Aton and E. D. Herzog, "Come together, right… now: synchronization of rhythms in a mammalian circadian clock," *Neuron*, vol. 48, no. 4, pp. 531–534, 2005.

[17] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2014.

[18] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 3, pp. 535–547, 2000. [Online]. Available: https://doi.org/10.1109/49.840210

[19] "The impact of asynchrony on stability of mac," 2024. [Online]. Available: https://www.dropbox.com/scl/fi/9o5294nkphky72wk5c23j/Asynchronous_Radio_Network.pdf?rlkey=8n54zibv14cxb1e217ds94vo8&dl=0

[20] J. Capetanakis, "Tree algorithms for packet broadcast channels," *IEEE Trans. Inf. Theor.*, vol. 25, no. 5, pp. 505–515, Sep. 1979. [Online]. Available: https://doi.org/10.1109/TIT.1979.1056093

[21] B. S. Chlebus, L. Gasieniec, A. Gibbons, A. Pelc, and W. Rytter, "Deterministic broadcasting in unknown radio networks," in *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*, D. B. Shmoys, Ed. ACM/SIAM, 2000, pp. 861–870. [Online]. Available: http://dl.acm.org/citation.cfm?id=338219.338652

[22] A. E. F. Clementi, A. Monti, and R. Silvestri, "Selective families, superimposed codes, and broadcasting on unknown radio networks," in *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA*, S. R. Kosaraju, Ed. ACM/SIAM, 2001, pp. 709–718. [Online]. Available: http://dl.acm.org/citation.cfm?id=365411.365756

[23] 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," *J. ACM*, vol. 32, no. 3, pp. 589–596, 1985. [Online]. Available: https://doi.org/10.1145/3828.214125

[24] D. Davis and S. Gronemeyer, "Performance of slotted aloha random access with delay capture and randomized time of arrival," *IEEE Transactions on Communications*, vol. 28, no. 5, pp. 703–710, 1980.

[25] A. Cohen, J. Heller, and A. Viterbi, "A new coding technique for asynchronous multiple access communication," *IEEE Transactions on Communication Technology*, vol. 19, no. 5, pp. 849–855, 1971.

[26] T. Cover, R. McEliece, and E. Posner, "Asynchronous multiple-access channel capacity," *IEEE Transactions on Information Theory*, vol. 27, no. 4, pp. 409–413, 1981.

[27] M. Yemini, A. Somekh-Baruch, and A. Leshem, "On the multiple access channel with asynchronous cognition," *IEEE Transactions on Information Theory*, vol. 62, no. 10, pp. 5643–5663, 2016.

[28] Z. Zhang, M. Yuksel, G. M. Guvensen, and H. Yanikomeroglu, "Capacity region of asynchronous multiple access channels with ftn," *arXiv:2301.02334*, 2023.

[29] S. Shahi, D. Tuninetti, and N. Devroye, "The strongly asynchronous massive access channel," *Entropy*, vol. 25, no. 1, 2023. [Online]. Available: https://www.mdpi.com/1099-4300/25/1/65

[30] B. S. Chlebus and M. A. Rokicki, "Centralized asynchronous broadcast in radio networks," *Theoretical Computer Science*, vol. 383, no. 1, pp. 5–22, 2007, structural Information and Communication Complexity (SIROCCO 2004). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397507002617

[31] G. De Marco and G. Stachowiak, "Asynchronous shared channel," in *Proceedings of the ACM Symposium on Principles of Distributed Computing*, ser. PODC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 391–400. [Online]. Available: https://doi.org/10.1145/3087801.3087831

[32] 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," *J. ACM*, vol. 48, no. 1, pp. 39–69, 2001. [Online]. Available: https://doi.org/10.1145/363647.363677

[33] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.