# Local Queuing Under Contention

**Paweł Garncarek**
Institute of Computer Science, University of Wroclaw, Poland
pgarn@cs.uni.wroc.pl

**Tomasz Jurdziński**
Institute of Computer Science, University of Wroclaw, Poland
tju@cs.uni.wroc.pl

**Dariusz R. Kowalski**
Computer Science Department, University of Liverpool, Liverpool, UK
D.Kowalski@liverpool.ac.uk

## Abstract

We study stability of local packet scheduling policies in a distributed system of $n$ nodes. The local policies at nodes may only access their local queues, and have no other feedback from the underlying distributed system. The packets arrive at queues according to arrival patterns controlled by an adversary restricted only by injection rate $\rho$ and burstiness $b$. In this work, we assume that the underlying distributed system is a shared channel, in which in order to get rid of a packet from the queue, a node needs to schedule it for transmission on the channel and no other packet is scheduled for transmission at the same time. We show that there is a local *adaptive* scheduling policy with relatively small memory, which is universally stable on a shared channel, that is, it has bounded queues for any $\rho < 1$ and $b \geq 0$. On the other hand, without memory the maximal stable injection rate is $O(1/\log n)$. We show a local memoryless (*non-adaptive*) scheduling policy based on novel idea of ultra strong selectors which is stable for slightly smaller injection $c/\log^2 n$, for some constant $c > 0$.

**2012 ACM Subject Classification** Theory of computation → Distributed algorithms

**Keywords and phrases** Distributed algorithms, local queuing, shared channel, multiple-access channel, adversarial packet arrivals, stability, deterministic algorithms

**Digital Object Identifier** 10.4230/LIPIcs.DISC.2018.28

**Funding** Supported by Polish National Science Center grant 2017/25/B/ST6/02010.

## 1 Introduction

Queuing processes have been in the heart of computing and communication for decades. Queues are governed by scheduling algorithms, and the desired property is stability – meaning that there is an upper bound on the numbers of packets queued at devices at any time. Recently, due to rapidly growing number of devices and popularity of distributed protocols, the impact of congestion on stability of queuing processes has become a vibrant research topic. In this work, we study stability of local packet scheduling policies in the process of dynamic broadcasting on a shared channel. A shared channel, also called a multiple access channel, is a broadcast network with instantaneous delivery of transmitted messages to every device (also called a node or a station) in the system and a possibility of conflict for access to the transmitting medium. A message sent via a channel by a station is received successfully by all the stations when its transmission has not overlapped with transmissions by other stations.


© Paweł Garncarek, Tomasz Jurdziński, and Dariusz R. Kowalski;
licensed under Creative Commons License CC-BY

The traditional approach to modeling dynamic broadcasting and the corresponding queuing process on a shared channel has assumed continuous packet injection subject to stochastic constraints (typically, Poisson arrival rates). Recent papers, inspired by adversarial queuing theory for store-and-forward packet networks, studied stability of deterministic broadcasting on a shared channel in adversarial settings. An adversary is determined by two parameters: injection rate $\rho$, which is the average number of injected packets, and burstiness $b$, which is the maximum number of packets that may be injected in a round. To the best of our knowledge, all previous work assumed that scheduling policies receive some feedback from the channel that help them in achieving stability. Our approach is the first that considers local schedulers without any feedback from the channel.

## 1.1 Our results

This paper investigates stability and other properties of deterministic local packet schedulers which are used for distributed broadcasting on a shared channel. The stability is studied in adversarial setting, defined in terms of global injection rate $\rho$ and burstiness $b$. We consider two classes of local schedulers: adaptive and non adaptive. The former allows stations to monitor and store some digest of the local queue history, especially its size, while the latter allows the policy only to check whether the current local queue is empty or not. We show that there is a local adaptive scheduling policy with relatively small memory, which is universally stable on a shared channel, that is, it has bounded queues for any $\rho < 1$ and $b \geq 0$. On the other hand, we prove that without memory the maximal stable injection rate is $O(1/\log n)$, more precisely, than any local non-adaptive scheduler can be stable only for injection rates $O(1/\log n)$. We also show a local non-adaptive policy based on novel idea of ultra strong selectors, which is stable for slightly smaller injection rate $c/\log^2 n$, for some constant $c > 0$.

## 1.2 Previous and related work

There is a rich history of research on broadcasting dynamically generated packets on multiple access channels. Early work in this direction included developing and studying properties of protocols like Aloha [1] and binary exponential backoff [23]. Most prior research on this topic has concentrated on scenarios when packets were injected subject to statistical constraints. Stability has been the basic quality criterion to achieve, understood in the sense that the input and output rates were equal. See the paper by Gallager [12] for an overview of early research; recent work includes the papers of Goldberg et al. [14], Goldberg et al. [15], Håstad et al. [17], and Raghavan and Upfal [24].

Adversarial queuing was introduced by Borodin et al. [8] as a framework to study stability of routing protocols in (point-to-point connected) networks under worst-case traffic scenarios modeled by adversaries. Andrews et al. [5] defined a greedy protocol to be universally stable when it was stable in all networks for any injection rate $\rho < 1$.

Bender et al. [6] studied stability of randomized backoff on a shared channel in adversarial settings in the *queue-free model* in which each packet is handled independently as if by a dedicated station. Stability was defined to mean that output rate was as large as injection rate. Some aspects of dynamic selection on multiple access channels by deterministic protocols were considered by Kowalski [21]. Both above approaches differ from the one in this paper, as they did not consider individual local queues.

Chlebus at al. [10] were the first who studied adversarial queuing on a shared channel. They however, similarly to all the follow up work (cf., [4, 3, 9]), assumed that schedulers are embedded into the channel, in the sense that they can receive channel feedback or

even attach and read additional information bits. Two of their results, however, could be transferred to the model of local scheduler assumed in this paper. Mainly, they showed that no acknowledgment-based protocol can be stable for injection rates larger than $\frac{3}{1+\lg n}$, and that there is a stable acknowledgment-based protocol for rates at most $\frac{1}{cn\lg^2 n}$, for a sufficiently large $c > 0$. This class of acknowledgment-based protocols is however different from the classes considered in this work, in the sense that the non-adaptive protocol restarts each time a new packet is considered by a scheduler – as such, it is difficult to compare their results with the ones in our work, and the only note we could make here is the huge exponential gap between the lower and upper bounds on stability rates.

Broadcasting on a shared channel with static input (i.e., when some $k$ stations hold packets at the beginning of an execution) was also widely studied. The goal is to transmit either at least one of them, which is the *selection problem*, or all of them in the *$k$-broadcast problem*, in both cases as quickly as possible. Kushilevitz and Mansour [22] proved the lower bound $\Omega(\log n)$ for the selection problem on the channel with $n$ stations if collision detection was not available. Willard [25] developed protocols solving the selection problem in the expected time $\mathcal{O}(\log\log n)$ in the channel with collision detection. This demonstrates that there is an exponential gap between models with collision detection and without it, with respect to the selection problem. The $k$-broadcast problem was studied by Greenberg and Winograd [16], Komlós and Greenberg [20], and Kowalski [21]. A related leader election problem was studied by Jurdziński et al. [18] for channels without collision detection. The related problem of wake-up, where stations become activated at possibly different times and have to transmit or exchange information, was also considered in the literature. It was introduced by Gąsieniec et al. [13]. They showed that if stations had access to a global clock then wakeup could be completed in the expected time $\mathcal{O}(\log n)$ by a randomized protocol; this was later strengthened by Jurdziński and Stachowiak [19] who showed that the assumption about the global clock could be omitted. Czyżowicz et al. [11] studied deterministic solutions for the mutual exclusion and consensus problems on multiple-access channels when the adversary wakes up stations in arbitrary rounds. They considered various model settings, determined by the availability of three independent features: collision detection, global clock and knowledge of the number $n$ of stations. In particular, they showed that if none of the three features is available then the problems are not solvable, while even a single of these features makes these problems feasible, with complexity of solutions depending on which combination of features is available. Bieńkowski et al. [7] analyzed randomized protocols for mutual exclusion under the model settings of [11].

## 2 Model

There are $n$ stations attached to a shared channel. The stations have distinct names in the interval $[0, n-1]$. Each station $v$ knows its name $v$ and also the number of all the stations $n$, in the sense that these parameters can be used in code of protocols.

### 2.1 Shared channel

Shared channel, also called a multiple access channel, is a communication medium with some special properties. We assume that the channel operates synchronously. Every device connected to it, called a node or a station, has its clock and the clock cycles are all of exactly the same length and synchronized. An execution of a protocol is partitioned into rounds – it takes precisely one round to transmit a message. We assume that stations have access to a global clock, meaning that all the local clocks at stations give the same round numbers.

Every station occasionally receives packets to broadcast. Packets are stored in a local queue. Stations use local scheduling algorithms to decide whether to schedule a packet from the queue for transmission in the current round or not. When exactly one station schedules a packet for transmission in a round, then the message containing this packet is successfully delivered and the packet disappears from the queue. Unlike in the previous work, we assume that local schedulers do not receive any feedback from the channel – they could only make their decisions based on examining local queue. When at least two stations transmit simultaneously in a round then *conflict for access* or *collision* occurs in the round and none of the transmitted packet is successful (i.e., all remain in their local queues).

## 2.2 Adversaries injecting packets

Packets are injected into stations in a dynamic fashion in the course of an execution of a broadcasting protocol. Packet injection is modeled by an adversary. Adversaries are specified by constraints on the maximum rate of injection $\rho$ and by the burstiness of traffic $b$. An adversary generates a number of packets in each round and for each packet assigns a station to inject the packet in this round. The number of packets an adversary can inject into stations in one round is called the *burstiness* of the adversary. The adversary of type $(\rho, b)$ can inject at most $\rho t + b$ packets to stations, in total, during any time interval of $t$ rounds. This type of adversary is typically called *leaky bucket*.

## 2.3 Local schedulers

A broadcast algorithm is determined by the work of local scheduling algorithms at each station. A local scheduler is a deterministic and distributed algorithm, which has only access to the current local queue, global clock, and potentially some additional internal memory (different from the local queue), but *does not receive* any feedback from the channel. It decides whether to schedule a packet for transmission in the current round or not. A packet scheduled for transmission is removed from the queue automatically if the transmission was successful (i.e., no other packet was scheduled for transmission in the same round), and the scheduler is aware of it.

W.l.o.g. we assume that the queue at a station operates in the first-in-first-out (FIFO) fashion, as we are only interested in stability (i.e., bounded queue sizes).

We focus on two classes of local schedulers. First class, *adaptive schedulers*, allows the algorithm to access the current queue (in particular, knows its size) and can use additional local memory (different from the local queue) to store some digest of queue history. The second class, *non-adaptive schedulers*, requires that the scheduler knows only if the current queue is empty or not and does not use local memory to record *any* digest of the history of local computation.

## 2.4 Quality of service

We say that a local scheduler is *stable* for injection rate $\rho$ if in any execution of the scheduler against a $(\rho, b)$-adversary, for any $b$, queues are bounded at all times. *Universal stability* means stability for any injection rate $\rho < 1$.

# 3 Universally stable algorithm with (small) memory

In this section we build a universally stable algorithm under the assumption that nodes can collect information about history of an execution in their local memory.

We construct the algorithm gradually: first, we describe the basic window-base brute-force idea for known injection rate $0 < \rho < 1$ and burstiness $b \geq 0$ and under assumption that a node has an additional feedback from the channel; second, we modify it to get rid of the above mentioned assumptions.

## 3.1 Window-based algorithm under additional assumptions

As mentioned earlier, the first algorithm will be designed for known injection rate $\rho$ and burstiness $b$. Additionally, in this part we only consider the beeping channel feedback model, where a node can distinguish between no transmitting nodes and at least one transmitting node on the channel; in other words, a node can use so called "beeps" (i.e., at least one transmission on the channel) to encode some control information to other nodes.

The algorithm partitions time into windows of length $L$. Each window consists of a *gossip stage* and a *transmission stage*. In the gossip stage, the nodes learn the queue sizes of other nodes. Then they use this knowledge to compute locally an optimal schedule – as we will argue later, it will be the same at each node. In the transmission phase, the nodes use the schedule they computed in the preceding gossip phase to transmit all packets that were injected in the previous window. Below we specify the two stages in details.

### Gossip phase

Let $f(L) = \log L$ denote the maximal number of bits that any node must exchange in a gossiping algorithm in order to share its queue size taken from the beginning of the window (i.e., the size just at the beginning of the window); if the queue size is greater than $L$, the node only announces it has $L$ packets. A node may require $f(L)$ packets just to transmit the information in the gossip phase. Therefore each node actively participates in communication within a window only if it has at least $f(L)$ packets at the beginning of the window. Round robin algorithm is used for this purpose, repeated $f(L)$ times – a node transmits a packet only if it actively participates in the current window *and* it is its turn on the list of stations used by the round robin algorithm *and* it has 1 on the $i$-th position of the binary representation of its queue size from the beginning of the window, where $i$ is the current number of iterations of round robin. In order to perform this algorithm, memory size $f(L)$ is required at each node.

### Transmission phase

Each node has full knowledge about the queue states of all active nodes in the system (subject to restrictions that for queues larger than $L$ the value $L$ is known) at the beginning of the transmission phase. Note that memory size $n \cdot f(L)$ is needed for it. Therefore all active nodes can compute the same transmission schedule to be used in the remainder of the window. The nodes use a brute force algorithm to find the shortest schedule that transmits all packets injected before the end of the previous window and not yet transmitted, and then follow this schedule. There are at most $\rho L + b$ packets injected into the system in the previous window, therefore a window of length $T(\rho, b) + G(n, L)$ is long enough, where $T(\rho, b) = \rho L + b$ is the length of the transmission phase, while $G(n, L)$ is the length of the gossiping phase (in the gossiping phase the propagated queue sizes are capped by $L$, hence $\log L$ bits are enough in

**Algorithm 1** Local Scheduling with Memory for a station $v$.

```
1: L_v ← min_length
2: loop
3:     old_v ← number of packets in the queue
4:     q_v ← number of packets injected in the previous window
5:     if old_v > f(n, L_v) then
6:         (schedule, increase?) ← Gossip(L_v)
7:         if increase? then
8:             L_v ← 2L_v
9:             Idle until round t such that t is divisible by the new L_v
10:        else
11:            Transmit(schedule)
```

each such transmission). Therefore, if we use window length $L \geq (b + G(n, L))/(1 - \rho)$, there is enough time to transmit all packets from the previous window. Indeed, for such values of $L$ we have the desired property $L \geq T(\rho, b) + G(n, L)$.

Since we iterate round robin algorithm $f(L)$ times for gossiping, we get $G(n, L) = n \log L$. Therefore, there exists a sufficiently large window length $L$ such that $L \geq (b + n \log L)/(1 - \rho)$ and consequently the above algorithm transmits all packets injected in a window during the next window from all active nodes. This proves (recursively) that all packets in queues at the beginning of a window have been injected during the previous window (except for at most $f(n, L)$ packets in each station, which could be injected earlier without activating the station), automatically implying stability.

## 3.2 Modified algorithm

The above brute-force window-based algorithm has a number of weaknesses:

1. $\rho$ and $b$ are known
2. users need to listen to the channel feedback (as opposed to only detecting own collisions)

Next we modify the above algorithm to get rid of these weaknesses. A high-level description of algorithm executed by a node is presented below, as Algorithm 1., with subprocedures Gossip (Procedure 2), Exchange (Procedure 3) and Transmit (described only in words).

Each node $v$ knows its previous window size $L_v$ (different nodes may have different window lengths). At the start of the algorithm, each node starts with a window of length $min\_length = n(n-1) \cdot 8$. To minimize desynchronization that occurs, a node $v$ only starts a window of length $L_v$ if the current round $t$ is divisible by $L_v$.

Only nodes that started their window with at least $n(n-1) \cdot \log L_v$ packets in their queues participate in the window – these are called *active nodes*. Each inactive node $u$ idles until it gets enough packets to become active (and then it idles until such a round $t$ that is divisible by $L_u$).

In a window all active nodes learn eachother's *fresh packets* – packets injected in the previous window – and decide on a common transmission schedule that transmits all fresh packets. If the computed schedule is too long to fit by the end of their window, each active node $v$ doubles its window size (and idles until the next window starts). Otherwise they transmit according to this schedule. If an active node $v$ learns that some other active node $u$ uses a shorter window than $L_v$, then $v$ informs $u$ about this and $u$ doubles its window length (and idles until the next window starts).

**Algorithm 2** Procedure $Gossip(L_v)$.

```
1: ignore ← Exchange(1⃗)
2: ignore ← Exchange(1⃗)
3: die? ← Exchange(0⃗)
4: if die? ≠ 0⃗ then
5:     Idle for 3(⌈log W_v⌉ + 1) phases
6:     return (null, true)
7: q_1, q_2, ..., q_⌈log W_v⌉ ← bits of min(q_v, 2^⌈log W_v⌉)
8: for i ← 1; i ≤ ⌈log W_v⌉; ++i do
9:     if q_i = 1 then
10:        x⃗ ← Exchange(1⃗)
11:        y⃗ ← Exchange(0⃗)
12:    else
13:        x⃗ ← Exchange(0⃗)
14:        y⃗ ← Exchange(1⃗)
15:    for j ← 1; j ≤ n; ++j do
16:        if x_j = y_j then
17:            z_j ← 1
18:        else
19:            z_j ← 0
20:    die? ← Exchange(z⃗)
21:    if die? ≠ 0⃗ then
22:        Idle for 3(⌈log W_v⌉ + 1 − j) phases
23:        return (null, true)
24: ignore ← Exchange(0⃗)
25: ignore ← Exchange(0⃗)
26: die? ← Exchange(0⃗)
27: if die? ≠ 0⃗ then
28:    return (null, true)
29: schedule ← compute(exchange_results)
30: return (schedule, false)
```

**Exchange subprocedure.** Exchange subprocedure's goal is to transmit a single bit of information to and from each active node (it can transfer different bits to each node) and therefore implements *beeps* from the previous section.

It takes $n(n-1)$ rounds. This timeframe is divided into $n$ intervals of $n-1$ rounds each. In the $i$-th interval a $i$-th node has a chance to transmit a bit to each of the $n-1$ other nodes. $i$-th node transmitting a packet in $j$-th round of $i$-th interval means that $i$-th node rties to transmit a binary signal 1 to the $j$-th node (to the $j+1$-th node if $j \geq i$). No transmission in $j$-th round of interval $i$ from node $i$ means that either $i$-th node transmits a binary 0 to the $j$-th node (or $j+1$-th node) or it is inactive.

Furthermore, the active nodes need to be able to receive these bits of information. Each active node $j$ transmits a packet in $j$-th round of every interval $i$ for $i < j$ and of every interval $i+1$ for $i \geq j$. If a collision occured in $j$-th round of interval $i$, then either (for $j < i$) $j$-th node interprets this as $i$-th node transmitting a bit 1 or (for $j \geq i$) $j$-th node interprets this as $i+1$-th node transmitting a bit 1.

**Algorithm 3** Procedure $Exchange(\vec{x})$ for a node $v$.

```
1:  for block ← 1; block ≤ n; ++ block do
2:      if block = v then
3:          for j ← 1; j ≤ n − 1; ++ j do
4:              if x_j = 1 then
5:                  Send a packet
6:              else
7:                  Idle for 1 round
8:      else
9:          for j ← 1; j ≤ n − 1; ++ j do
10:             if (v < block and v = j) or (v > block and v = j + 1) then
11:                 Send a packet
12:             else
13:                 Idle for 1 round
```

Note that the *exchange* procedure has no guarantees about successful transmissions. It is used solely for communication.

**Gossip stage.** Gossip stage is used to share queue sizes of al active nodes, so that they can agree on a common schedule. Furthermore it is used to find out whether some nodes should increase their window sizes.

Gossip stage is divided into $\log L_v + 2$ *phases*. Each phase consists of 3 *exchange* routines. The first phase and the last phase are used to inform other gossiping nodes about $v$'s start and end of gossiping stage. The middle $\log L_v$ phases are used to transmit $v$'s queue size, each phase for one bit.

The first phase of gossip of node $v$ is always exchanging 110 to each other node. The last phase of gossip is always exchanging 000 to each other node. In the middle phases, a transmission of a bit 1 is encoded as exchanging 10? to each other node, while a transmission of a bit 0 is encoded as exchanging 01? to each other node, where the third exchange of these phases will be described later.

This kind of coding is used to ensure that nodes with different window lengths, and therefore desynchronized gossiping phases, find out about eachother and learn which one has shorter window length. Indeed, if another node $u$ with a shorter window length starts a gossip stage at the same time as $v$, $v$ learns that $u$ has a shorter gossip (and therefore a shorter window $L_u$) after 2 exchanges of the last $u$'s phase (00 are enough to identify the finishing phase). If $u$ starts a gossip during $v$'s gossip stage, within 2 exchanges (11 at the start of $u$'s gossip) $v$ learns about it.

For every node $u$ that starts of finishes a gossip during $v$'s gossip, $v$ exchanges *kill* to node $u$ – a bit 1 on the third exchange of the phase when $v$ found out that $u$ is desynchronized (this was denoted earlier by a '?'. A node that hears a kill from any other node immediately stops its transmissions, doubles its window size and waits until the next start of a window.

If the nodes after a gossiping phase find out that they are not able to transmit all packets from the previous window (i.e., the computed schedule would be too long for the allocated length of transmitting stage), then they double their window lengths.

**Transmission stage.** Transmission stage is used to transmit all fresh packets, according to a schedule found during a gossip stage. Note that gossip stage may successfuly transmit some fresh packets or packets injected in earlier windows. Therefore a node may have not enough fresh packets to use the schedule, but that means that in this window it successfuly transmitted all fresh packets.

Note that the schedule guarantees no collisions between nodes that started their windows at the same time. However, a collision may occur with a node which started its window at a different time, so with a different window length. If a node $v$ detects a collision during its transmission stage, it stops following its schedule, and instead transmits kill messages to all other nodes until it runs out of packets or the window ends. These kill messages are transmitted in all rounds during intervals $I = [t + 2n(n-1), t + 3n(n-1))$, for every $t$ divisible by $3n(n-1)$ (these are the rounds reserved for kill messages in gossiping nodes).

These transmissions will cause collisions with other active nodes $u$ in their transmission stage or in their gossip stage. So $u$ will either start transmitting kill messages (in case it happened during the transmission stage), or it will interpret this collision as a kill message (if it happened during the gossip stage).

## 3.3 Analysis of the modified algorithm

▶ **Theorem 1.** *There exists an adaptive scheduler against any adversary with injection rate $\rho < 1$ and burstiness $b$.*

We show that the algorithm described above is a scheduler satisfying requirements of the theorem. We need to prove 2 lemmas:

▶ **Lemma 2.** *For any $\rho$ and $b$ there exists a window length $L_{max}$, such that it is never exceeded.*

▶ **Lemma 3.** *In every window of length $L_{max}$ either all nodes transmit all packets injected in the previous window of length $L_{max}$ or some node decides to increase its window length.*

If both lemmas hold, then there is a finite number of windows where packets are processed slower than they are injected. Therefore the queues are bounded and the algorithm is universally stable. It remains to show that both above lemmas hold.

**Proof of Lemma 2.** For every $n$, $\rho$, $b$ there exists a window length $L$ such that if every node achieves it, all the nodes transmit all the packets injected in the previous window, without collisions during transmission stages (see the explanation in the base algorithm, Section 3). Observe that a node $v$ increases its window length in two cases:

- Case 1: During *Gossip* stage it heard a *kill* message from another node $u$.
- Case 2: During *Gossip* phase it found out that it has too many packets to transmit in a window of current length.

Note that the 1st case can only occur if $u$ has a longer window than $v$, therefore $L_v < L$, so $2L_v \leq L$.

The 2nd case allows a node with the longest window in the entire network to increase its window length. This however happens only if the current window length is insufficient and therefore shorter than $L$.

Therefore the lemma holds for $L_{max} = L$. ◀

**Proof of Lemma 3.** If a node is unable to transmit all fresh packets, then in the gossip phase it doubles its window length. If a node $v$ was unable to successfully transmit all fresh packets due to collisions during $v$'s transmissions stage with a node $u$ that has a different window length, one of the cases below occured:

1. $L_u < L_v$ and $u$ started its window before (or in the same phase) $v$ finished its gossip stage – $u$ receives a kill from $v$ at the start of $u$'s gossip.
2. $L_u < L_v$ and $u$ started its window after $v$ finished its gossip stage – $v$ detects a collision during its transmission stage and starts transmitting kill messages instead of following its schedule. Either $u$ heard a kill message from $v$ or $v$ ran out of packets (transmitted all fresh packets).
3. $L_u > L_v$ and $u$ started its window at the same time that $v$ – $v$ receives a kill at the end of its gossip stage.
4. $L_u > L_v$ and $u$ started its window before $v$ and $u$ finished its gossip stage after (or in the same phase) $v$ started its gossip stage – $v$ receives a kill at the start of its gossip
5. $L_u > L_v$ and $u$ started its window before $v$ and $u$ finished its gossip stage before $v$ started its gossip stage – $u$ detects a collision during its transmission stage and starts transmitting kill messages instead of following its schedule. Either $v$ heard a kill message from $u$ or $u$ ran out of packets (transmitted all fresh packets).

Therefore any active node $v$ during its window either:

- transmits all its fresh packets,
- finds out its window length is too short to transmit its packets,
- receives a kill message,
- transmits a kill message that is received by another node.

So either all active nodes transmit all their fresh packets in their every window during interval of length $L_{max}$ or there exists a node that increased its window length. ◀

**Memory usage.** Note that a node remembers queue sizes of all other nodes, where queue sizes are bounded by $L_{max}$ (a value dependent on $b$, $\rho$ and $n$). So each node uses at most $n \cdot \log L_{max}$ bits of memory.

# 4 Unknown queue size

In this section we consider a class of protocols where stations do not know how many packets they have themselves. Each station decides whether to send a packet or not based only on the current round number. If such station has no packets to send, it simply fails.

## 4.1 Impossibility result

In this subsection we will show that no oblivious algorithm can be stable against an adversary with injection rate $\rho = \omega(1/\log n)$. Firstly, we introduce some auxiliary terminology and provide a high level description of the proof of the result. Then the formal proof follows.

### 4.1.1 Auxiliary terminology and high level description of the impossibility result

For a given set $X$, let $2^X$ denote the set of all subsets of $X$.

▶ **Definition 4.** A *scenario* is a class of packet arrival patterns where only a specific subset of stations have packets injected into them.

According to the above definition, each scenario can be identified with a subset of $[n]$ corresponding to nodes to which packets can be injected. And therefore, there are $2^n$ possible scenarios.

Consider a fixed oblivious algorithm $A$ for a system which consists of $n > 0$ nodes. Then, for a fixed set of scenarios $S \subset 2^{[n]}$ (i.e., subsets of $U = [n]$), let us build a bipartite graph $G(U \cup S, E)$, where $(u, s) \in E$ iff $u \in s$. That is, $(u, s)$ denotes the fact that an adversary can inject packets to the node $u$ in the scenario $s$. [Observe that each bipartite graphs with partition of nodes $U \cup S$ and $U = [n]$ uniquely describes a set of scenarios and each set of scenarios $S$ uniquely determines the graph $G(U \cup S, E)$.]

We say that an execution of a system is *saturated* with respect to a scenario $s \subseteq [n]$ in some period of time when each queue $i \in s$ is nonempty at each round of the given period.

For a given schedule $A$, let us restrict to its saturated executions on some set of scenarios. Then, one can characterize succesful rounds of schedule $A$ (i.e., such that a packet is transmitted) in saturated executions for various scenarios, using the following terminology.

We say that a set $H \subseteq [n]$ *hits* a scenario $s \subseteq [n]$ iff $|H \cap s| = 1$. Now, for a given round $t$, let $A_t \subseteq [n]$ be the set of nodes which transmit a message in round $t$ according to the schedule $A$ (provided their queues are not empty). We also say that the scenario $s$ is *successful* in round $t$ of $A$ if a packet is successfully transmitted in round $t$ of a saturated execution of $A$ in the scenario $s$. One can easily observe that $s$ is successful in round $t$ of $A$ iff $A_t$ hits $s$, i.e., $|A_t \cap s| = 1$. Thus, the number of (saturated) executions from a given set $S$ in which a packet is successfully transmitted in round $t$ is equal to the number of elements of $S$ which are hit by $A_t$.

Interestingly, the above characterization of successful transmissions in saturated executions in the scenarios from the set $S$ is closely related to broadcasting in radio networks. Let us consider a restricted broadcasting problem, where all nodes from $U$ know a message $\mathcal{M}$ and the goal is to deliver $\mathcal{M}$ to all nodes from $S$ in a radio network described by $G(U \cup S, E)$. The radio network [2] is the model of distributed networks, where node $u$ of a graph $G$ receives a message from $v$ in a round $t$ iff $v$ is the only (in)neighbor of $u$ in $G$ transmitting a message in $r$. One can observe that, for the set of transmitters $A_t$,

- the number of successful scenarios from $S$ in round $t$,
- the number of elements of $S$ which receive a message (from nodes in $U$) in the radio network $G(U \cup S, E)$ in a round in which the set of transmitters is equal to $A_t$, and
- the number of elements of $S$ hit by $A_t$

coincide. Using this relationship, we will be able to use the lower bound for broadcasting in radio networks from [2], which in particular implies that the above restricted broadcasting problem requires $\Omega(\log^2 n)$ rounds.

In the nutshell, the proof of our impossibility goes as follows. Firstly, we focus merely on saturated executions. For a given set of scenarios $S$, we say that a round $t$ is *c-sparse* if the number of successful scenarios from $S$ in $t$ wrt saturated executions is at most $O(|S|/(c \log |S|))$. (Recall that the set of successful scenarios from $S$ in round $t$ is equal to the set of those elements of $S$ which are hit by the set of transmitters $A_t$.) A set of scenarios $S$ is *universally c-sparse* if it is $c$-sparse for each possible round of each schedule (i.e., for each possible set of nodes transmitting in a round).

The main technical step towards our result is phrased in the lemma (Lemma 8) which says that, for some constant $c_2 \geq 1$ and for some arbitrarily large $n$, there exists a universally $c_2$-sparse set of scenarios of size polynomial wrt $n$. Intuitively, this technical claim says that only a fraction of $\frac{1}{c_2 \log |S|}$ scenarios from $S$ are successful in each round of each oblivious schedule $A$. We prove this particular lemma by contradiction: assuming that it is false, we build a schedule for the restricted broadcasting problem in radio networks of length $O(\log^2 n)$, which contradicts the lower bound for this problem from [2].

The above described technical result implies that, on average, a scenario from a certain set $S$ is successful once in each $c_2 \log |S|$ rounds. This in turn implies that, for each oblivious algorithm $A$ and each round $t$, there exists a scenario $v \in S$ and time period $[t, t+s)$ $(s \geq n)$ such that at most $s\rho'$ packets are successfully transmitted during a saturated execution of $A$ in the period $[t, t+s)$ for the scenario $v$ and some $\rho' = \frac{1}{c_2 \log |S|} = O(1/\log n)$.

However, the above reasoning holds only for saturated executions. In order to generalize this idea for arbitrary executions, we need to show that an adversary is able to inject packets in such a way that an actual executions worsk as an suturated one, for arbitrary long period of time. We deal with this challenge in Lemma 6, proved by contradiction.

### 4.1.2 Formal proof of the impossibility result

Let $n$ be the number of stations, $\rho = \delta/\log n$ for some constant $\delta$ be the injection rate of an adversary $ADV$ and $b > n$ be the burstiness of the adversary $ADV$. Let us fix an algorithm $ALG$. Let $f(x)$ be an arbitrary function such that $f(x) \to \infty$ if $x \to \infty$ and $f(x) = o(x)$.

▶ **Theorem 5.** *There exist arbitrarily large numbers of stations $n$ such that no non-adaptive scheduler can have bounded queues against an adversary with injection rate $\rho = 1/\log n$ and burstiness $b > n$.*

To prove Theorem 5, we will need additional definitions and Lemma 6.

Let $Alg(v, I)$ for some scenario $v$ and time interval $I$ be the number of packets successfuly transmitted by $ALG$ during interval $I$ under a saturated execution.

An interval $I = [t_b, t_e]$ is a *$\rho$-bounded interval* if there exists a scenario $v \in S$ and a constant $s^* = b - n \geq 1$ such that for every time $t \in I$:

$$Alg(v, [t_b, t)) - \rho' \cdot (t - t_b) < s^*.$$

▶ **Lemma 6.** *Consider $\rho' = c/\log n < \rho$. There exist arbitrarily large numbers of stations $n$ and an infinite sequence of time prefixes $\mathcal{T}_i$ such that for each prefix $\mathcal{T}_i$ there exists a $\rho'$-bounded interval $I \subseteq \mathcal{T}_i$ of length $f(\tau_i)$, where $\tau_i = |\mathcal{T}_i|$.*

**Proof of Theorem 5.** Consider an adversary $ADV'$ with injection rate $\rho' = O(1/\log n)$ and burstiness $b = n + s^*$. Let $\mathcal{T}$ be one of the prefixes $\mathcal{T}_i$ whose existence is postulated in Lemma 6 and $\tau = |\mathcal{T}|$. Let $v$ be the scenario and $I = [t_b; t_e)$ – the interval of length $f(\tau)$ that correspond to $\mathcal{T}$. $ADV'$ will wait without injecting any packets until $t_b$. Then in one round it will inject 1 packet to each station in scenario $v$ (using at most $n$ from its burstiness). After that (until $t_e$), whenever $ALG$ successfuly transmits a packet in the scenario $v$, $ADV'$ injects a new packet to the station that just transmitted. According to Lemma 6 $ADV'$ can do this using at most additional $s^*$ of its burstiness.

Therefore an adversary $ADV$ with injection rate $\rho = \omega(1/\log n)$ and burstiness $b = n + s^*$ can inject an additional $(\rho - \rho') \cdot f(\tau)$ packets into the system that will not be transmitted by time $t_e$. So as we pick longer prefixes $\mathcal{T}$ (that exist according to Lemma 6), we get $f(\tau) \to \infty$ and thus $(\rho - \rho') \cdot f(\tau) \to \infty$. So the queues are not bounded. ◀

Before we prove Lemma 6, we introduce a new structure and more auxiliary lemmas.

▶ **Definition 7.** A connected graph $G$ is $(n, c_2)$-good if:

1. $G$ is bipartite, with partition $U$ and $S$,
2. $|U| = n$,
3. $|S| = s = poly(n) \leq n^{c_2}$,
4. $|S| \geq |U|$,
5. each node in $S$ has at least one neighbour in $U$.

The set $U$ represents original stations. Each element $a \in S$ represents a scenario, where only the nodes adjacent to $a$ in graph $G$ receive packets.

▶ **Lemma 8.** *There exist arbitrarily large numbers $n$, such that there exists a $(n, c_2)$-good graph $G = (U \cup S, E)$ such that in one round $U$ can transmit a packet to at most $\frac{s}{c_2 \log s}$ nodes in $S$, where $s = |S|$ and $c_2 \geq 1$ is some constant.*

We will prove the above lemma later (but we use it now).

**Proof of Lemma 6.** Proof by contradiction.

Consider an adversary $ADV'$ with injection rate $\rho' < \rho$. According to Lemma 8 there exists arbitrarily large $n$ and a $(n, c_2)$-good graph $G$ such that in one round $U$ can transmit a packet to at most $\frac{s}{c_2 \log s}$ nodes in $S$.

Contrary, assume that for all scenarios $v \in S$, all intervals $I = [t_b, t_e] \subseteq \mathcal{T}$ such that $t_e - t_b = f(\tau)$, there exists $t \in I$ such that

$$Alg\,(v, [t_b, t)) - \rho' \cdot (t - t_b) \geq s^*.$$

Then we can split almost entire prefix $\mathcal{T}' = \mathcal{T} \setminus (\tau - f(\tau), \tau]$ into small intervals $J_1, J_2, \ldots, J_k$ for some $k$, where $J_i = [t_{i-1}, t_i)$ ends at time $t_i$ for which the above inequality holds, where $t_0 = 0$ is the beginning of prefix $\mathcal{T}$ and $t_k \in [\tau - f(\tau), \tau] \subseteq \mathcal{T}$. So for all $i$ we have $|J_i| \leq f(\tau)$ and $Alg(v, J_i) - \rho' \cdot (|J_i|) \geq s^*$. For each $J_i$ we have $Alg(v, J_i) \geq \rho' \cdot |J_i|$. We can sum this over all $J_i$ to get

$$Alg(v, \mathcal{T}') \geq \sum_i \rho' \cdot |J_i| \geq \rho' \cdot (\tau - f(\tau) + 1).$$

Let $B$ be the sum of all succesful transmissions by the algorithm $ALG$ across all considered scenarios $S$. So

$$B \geq \sum_{v \in S} ALG(v, \mathcal{T}') \geq s \cdot \rho' \cdot (\tau - f(\tau) + 1)$$

On the other hand, since the set of scenarios $S$ is represented by a $(n, c_2)$-good graph $G$, in at most $\frac{s}{c_2 \log s}$ scenarios a packet can be sent in one round of the algorithm (we picked our $n$ and $G$ according to Lemma 8). Therefore in $\tau$ rounds there are at most $\tau \cdot \frac{s}{c_2 \log s}$ packets sent across all scenarios. So

$$B \leq \tau \cdot \frac{s}{c_2 \log s}$$

Then we must have $\tau \cdot \frac{s}{c_2 \log s} \geq s \cdot \rho' \cdot (\tau - f(\tau) + 1)$, and thus $\rho' \leq \frac{\tau}{(\tau - f(\tau) + 1) \cdot c_2 \log s}$. Since $f(\tau) = o(\tau)$, for large enough $\tau$ we have $\frac{\tau}{(\tau - f(\tau) + 1)} < 2$ and so

$$\rho' \leq \frac{1}{c_2 \log s} = \frac{1}{c_2 \log n^{c_2}} = \frac{1}{c_2 c_2 \log n}$$

However $\rho' = c/\log n$ and for $c > 1/(c_2)^2$ we have a contradiction! ◀

**Proof of Lemma 8.** Proof by contradiction.

Suppose that for all $s$, for all $(n, c_2)$-good graphs $G = (U \cup S, E)$ where $|S| = n^{c_2} = s$, there exists a subset of transmitters $U$ that in one round can transmit a packet to more than $\frac{s}{h(n) \log s}$ receivers in $S$, where $h(n) = \omega(1)$ and $h(n) = o(\log n)$. We will build a

broadcasting algorithm for a graph $H = (V, F)$ with source $a$, where $V = U \cup S \cup \{a\}$ and $F = E \cup \{(a, v) | v \in U\}$ that requires $o(\log^2 n)$ rounds, which contradicts the result of Alon et al. [2].

**The broadcasting algorithm.** According to our assumption we can inform in one round $\frac{s}{h(n)\log s}$ nodes in $S$. Let us define new sets $S'$ and $U'$:

- set $S'$ consists of all uninformed nodes from $S$ (so $|S'| \leq s - \frac{s}{h(n)\log s}$),
- each node in $S'$ has at least one neighbour in $|U'|$ (for each node $v \in S'$ we take $v$'s arbitrary neighbour and add it to $U'$; so far $|U'| \leq |S'|$),
- if $|U'| \leq |S'|^{1/c_2}$ then we take arbitrary nodes $v \in U$ and add them to $U'$ until $|U'| = |S'|^{1/c_2}$ (note that $|S'|^{1/c_2} \leq |S'|$).

Let $G'$ be a subgraph of $G$ induced by nodes $U' \cup S'$. Let $s' = |S'|$ and $n' = |U'|$. Then the graph $G'$ is a $(n', c_2)$-good graph.

We repeat our algorithm recursively until the size of set $S$ is constant. Let $T(s)$ be the number of iterations required. $T(s) = 1 + T(s - \frac{s}{h(n)\log s})$. The value of $s$ will be halved after every $\frac{s/2 \cdot \log(s/2)}{h(s/2)}$ iterations. So $T(s) \leq \frac{s/2 \cdot \log(s/2)}{h(s/2)} + T(s/2)$. Thus $T(s) = o(\log^2 s)$. Each iteration takes only 1 round, so we only used $o(\log^2 s) = o(\log^2 n)$ rounds. ◀

## 4.2 Algorithm

In this section we show that there exists an oblivious algorithm which is stable for injection rates $\rho = O(\frac{1}{\log^2 n})$. Recall that we consider such protocols that the transmission pattern of each node is determined by ID of the node and the number of nodes $n$. That is, the pattern for a node is a 0-1 sequence.

▶ **Theorem 9.** *There exists a stable algorithm for injection rates $\rho = O(\frac{1}{\log^2 n})$, such that transmission pattern of each node is determined by its ID.*

We describe the proof of Theorem 9 in the remaining part of this section. More precisely, we show that a stable algorithm exists for each $n$. In the proof, we use Probabilistic Method.

The schedule for each ID $i \in [n]$ will be just a 0-1 sequence of length $W = O(n^2)$ repeated infinite number of times. (The exact value of $W$ will be determined later.)

Let $l = \log n$ (more precisely, $l = \lfloor \log n \rfloor$). For a given $W \in \mathbb{N}$, we build a 0-1 sequence $X_i$ of length $W$ for each ID $i \in [n]$ such that $\text{Prob}(X_i[t] = 1) = \frac{1}{2^{1+i \mod l}}$ for each $i \in [W]$ and all probabilistic choices are independent. Moreover, we split $W$ into consecutive phases of length $l$. That is, the first $l$ elements of $W$ form the first phase, the next $l$ elements of $W$ form the second phase, and so on. And, the number of phases is equal to $F = W/l = W/\log n$. Consider an execution of a schedule of lenght $W$ determined by the above defined random sequences under the assumption that the number of injected packets during an execution of a schedule is at most $\rho W$, where $\rho = O(1/\log^2 n)$. (The actual value of the constant hidden in the big-O notation will be determined later.) Thus, an adversary can inject packets in at most $\rho W = O\left(\frac{1}{\log^2 n} F \log n\right) = O(F/\log n)$ phases. Thus, no packet is injected in at least $F(1 - \frac{1}{\log n}) \geq \frac{1}{2}F$ phases. A phase in which

- no packet is injected, AND
- no queue becomes empty

is called a *clear phase*.

Now, let us consider a clear phase. Let $m$ be the number of nodes with non-empty queues at the beginning of such a phase. Moreover, let $k \leq l = \lfloor \log n \rfloor$ be such that $2^k < m \leq 2^{k+1}$. Consider two cases:

**(i)** no packets are successfully transmitted in the first $k-1$ rounds of the phase,

**(ii)** at least one packet is transmitted in the first $k-1$ rounds of the phase.

For (i), the probability that a packet is successfully transmitted in the $k$th round of the phase is $m \cdot \frac{1}{2^{k+1}} \cdot \left(1 - \frac{1}{2^{k+1}}\right)^{m-1} \geq \frac{1}{4e} = c$, where $e$ is the base for natural logarithms. In the case (ii) we are guaranteed that at least one packet is transmitted in a phase. Thus, the probability that (at least one) packet is transmitted in a clear phase is at least $c$, where $c$ is a constant.

As the number of clear phases is at least $\frac{1}{2}F$ and the probability of a successful transmission in a clear phase is $\geq c$, the expected number of successful transmissions in $W$ rounds (i.e., $F$ phases) is $\geq \frac{F}{2c} = \frac{W}{2c \log n} \geq 4\rho W$, provided $\rho = O(1/\log n)$ is sufficiently small. Significantly, as all the random choices are independent, we can use Chernoff bounds to estimate the probability that the actual number of transmitted packets is close to its expectation. More precisely, let $X$ denote the number of transmitted packets in the considered $W$ rounds. As we have shown, $X$ is bounded from below be the sum $X$ of $F/c$ independent 0-1 random variables such that $E[X] \geq \frac{F}{2c}$, where $c$ is a fixed constant. Thus,

$$\text{Prob}(X < 4\rho W) < \text{Prob}\left(X < \frac{F}{4c}\right) < \text{Prob}\left(X < (1 - \frac{1}{2})E[X]\right) < e^{-E[X]/8} = e^{-\frac{W}{16c \log n}}. \tag{1}$$

**Derandomization.** Now, our goal is to show that, with non-zero probability, a randomly chosen oblivious schedule of length $W$ (i.e., $n$ random 0-1 sequences of length $W$) guarantees $\geq 4\rho W$ successful transmissions irrespective of the injection pattern, provided the number of packets in queues at the beginning of an execution of this schedule is at least $\geq F/(4c) \geq 4\rho W$ and the number of injected packets is at most $\rho W$. By Probabilistic Method, this fact will imply that there exists an oblivious schedule which guarantees $4\rho W$ successful transmissions in $W$ rounds, provided there are at least $4\rho W$ packets at the beginning and at most $\rho W$ packets are injected during the considered period of $W$ rounds.

Let a *injection event* denote the event that a packet is injected in some round to some node. Moreover, let a *deletion event* denote the fact that the queue of a particular node becomes empty at particular round. Observe that the actual behaviour of a fixed schedule in a particular period $\mathcal{T}$ of $W$ rounds can be determined by the following factors:

- the set of nodes with non-empty queues at the beginning of $\mathcal{T}$,
- injections events during $\mathcal{T}$, where each injection event is described by: the ID $v$ of the node, the number of the round in which a packet is injected to $v$,
- deletion events during $\mathcal{T}$, where each deletion event is described by: the ID $v$ of the node and the number of the round in which the queue of $v$ becomes empty.

According to our assumptions, the number of injection events is at most $\rho W$. Each deletion event corresponds to a unique successful transmission of a packet. Thus, if the number of deletion events is larger than $4\rho W$, then the number of transmitted packets is also $\geq 4\rho W$. Therefore, it is sufficient to consider the case that the number of deletion events is $\leq 4\rho W$ and therefore the number of all events is at most $5\rho W$.

According to the above description of events, each event can be described in

$$\lceil \log n \rceil + \lceil \log W \rceil + 1 < 2(\log W + \log n) \leq 6 \log n$$

bits. This bound follows from the fact that $\lceil \log n \rceil$ bits are sufficient to encode ID, $\lceil \log W \rceil$ bits are sufficient to encode the round number in the schedule of length $W$ and one bit can encode the type of the event, either an injection event or a deletion event. (Let us stress here that we do not need the actual information about the sizes of queues at the beginning, since we assume that deletion events can be determined by an adversary. This assumption generalizes the scenario, where the queue becomes empty only after transmitting all packets in it.) Thus, finally, the size of the set of all possible scenarios is at most

$$2^n \cdot 2^{5\rho W \cdot 6 \log n} < 2^{31 \rho W \log n},$$

for large enough $n$, where

- $2^n$ is the number of possible sets of non-empty queues at the beginning;
- $2^{5\rho W \cdot 6 \log n}$ is the upper bound on the size of the set of possible events,
- the inequality follows from the fact that $W = n^2$ and $\rho = \Theta(1/\log^2 n)$.

As the probability that the number of transmitted packets is smaller than $4\rho W$ is at most $e^{-\frac{W}{16c \log n}}$ for a fixed scenario (see (1)), the probability that a random schedule does not guarantee at least $\frac{F}{4c}$ successful transmissions is at most

$$2^{31 \rho W \log n} \cdot e^{-\frac{W}{16c \log n}} < 2^{31 \rho W \log n - \frac{W}{16c \log n}} < 1,$$

provided $\rho = O(1/\log^2 n)$ is small enough. Thus, by Probabilistic Method, there exists a schedule $\mathcal{S}$ of length $W$ which guarantees that at least $\rho W$ packets are successfully transmitted provided that the number of packets at the beginning is at least $\rho W$ and the number of injected packets is at most $\rho W$.

In the above reasoning we assumed that at most $\rho W$ packets can be injected in $W$ rounds. This is very restrictive, since the adversary can actually inject $\rho W + b$ packets in $W$ rounds, where $b$ is the burstiness. Significantly, $b$ is unknown and therefore a schedule is independent of $b$. In order to take this circumstance into account, let us consider any fixed $b \in \mathbb{N}$. Let $\mathcal{S}$ be a schedule of length $W$ which guarantees that $4\rho W$ packets are transmitted during $\mathcal{S}$, provided there are at least $4\rho W$ packets in queues at the beginning. We will analyze the schedule $\mathcal{S}$ in consecutive *stages* of length $T = 2bW$. Thus, the schedule $\mathcal{S}$ is repeated $2b$ times in a stage. We will show that the number of packets in queues is smaller than $2T$ at the beginning of each stage which in turn implies that the number of packets in queues is smaller than $3T$ in each round. Consider two cases:

**(a)** The number of packets in queues at the beginning of a stage is at least $T$ and at most $2T$.

**(b)** The number of packets in queues at the beginning of a stage is smaller than $T$.

For (b) observe that the adversary can inject at most $\rho T + b < T$ packets in a stage and therefore the number of packets at the beginning of the next stage is at most $2T$. Thus, it remains to consider (a). We say that an execution of $\mathcal{S}$ is *safe* if the number of injected packets during that execution is at most $\rho W$. As an adversary can inject at most $\rho T + b = 2b \cdot \rho W + b$ packets in the stage, at least $b$ out of $2b$ executions of $\mathcal{S}$ are safe. As shown above, at least $4\rho W$ packets are transmitted during a safe execution of $\mathcal{S}$. Eventually, the number of packets transmitted in a stage of length $T$ is at least $b \cdot 4\rho W \geq 2\rho T > \rho T + b$. Thus, the number of successfully transmitted packets is the stage is larger than the number of injected packets and therefore the number of packets in all queues at the beginning of the next stage is at most $2T$.

## 5 Conclusions

In this work we investigated how stability of local schedulers run on a shared channel depends on their adaptivity. A natural research direction includes studying of other types of schedulers, as well as delivering more quantitative measurements of schedulers' quality (such as latency, queue sizes, local memory size, etc.). Schedulers could also be studied in the context of other related models with contention, such as SINR or dependency-graph models.

## References


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

2 Noga Alon, Amotz Bar-Noy, Nathan Linial, and David Peleg. A lower bound for radio broadcast. *Journal of Computer and System Sciences*, 43(2):290–298, 1991. doi:10.1016/0022-0000(91)90015-W.

3 L. Anantharamu, Bogdan S. Chlebus, and Mariusz A. Rokicki. Adversarial multiple access channel with individual injection rates. In *Proceedings of the 13th International Conference on Principles of Distributed Systems (OPODIS)*, LNCS 5923, pages 174–188. Springer-Verlag, 2009.

4 Lakshmi Anantharamu, Bogdan S. Chlebus, Dariusz R. Kowalski, and Mariusz A. Rokicki. Deterministic broadcast on multiple access channels. In *Proceedings of the 29th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–5, 2010.

5 Matthew Andrews, Baruch Awerbuch, Antonio Fernández, Frank Thomson Leighton, Zhiyong Liu, and Jon M. Kleinberg. Universal-stability results and performance bounds for greedy contention-resolution protocols. *Journal of the ACM*, 48(1):39–69, 2001.

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

7 Marcin Bieńkowski, Marek Klonowski, Mirosław Korzeniowski, and Dariusz R. Kowalski. Dynamic sharing of a multiple access channel. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 83–94, 2010.

8 Allan Borodin, Jon M. Kleinberg, Prabhakar Raghavan, Madhu Sudan, and David P. Williamson. Adversarial queuing theory. *Journal of the ACM*, 48(1):13–38, 2001.

9 Bogdan S. Chlebus, Dariusz R. Kowalski, and Mariusz A. Rokicki. Maximum throughput of multiple access channels in adversarial environments. *Distributed Computing*, 22(2):93–116, 2009.

10 Bogdan S. Chlebus, Dariusz R. Kowalski, and Mariusz A. Rokicki. Adversarial queuing on the multiple access channel. *ACM Transactions on Algorithms*, 8(1):5:1–5:31, 2012.

11 Jurek Czyżowicz, Leszek Gąsieniec, Dariusz R. Kowalski, and Andrzej Pelc. Consensus and mutual exclusion in a multiple access channel. In *Proceedings of the 23rd International Symposium on Distributed Computing (DISC)*, LNCS 5805, pages 512–526. Springer-Verlag, 2009.

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

13 Leszek Gąsieniec, Andrzej Pelc, and David Peleg. The wakeup problem in synchronous broadcast systems. *SIAM Journal on Discrete Mathematics*, 14(2):207–222, 2001.

14 Leslie Ann Goldberg, Mark Jerrum, Sampath Kannan, and Mike Paterson. A bound on the capacity of backoff and acknowledgment-based protocols. *SIAM Journal on Computing*, 33(2):313–331, 2004.

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

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

17 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.

18 Tomasz Jurdziński, Miroslaw Kutyłowski, and Jan Zatopiański. Efficient algorithms for leader election in radio networks. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 51–57, 2002.

19 Tomasz Jurdziński and Grzegorz Stachowiak. Probabilistic algorithms for the wakeup problem in single-hop radio networks. In *Proceedings of the 13th International Symposium on Algorithms and Computation (ISAAC)*, LNCS 2518, pages 535–549. Springer-Verlag, 2002.

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

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

22 Eyal Kushilevitz and Yishay Mansour. An $\Omega(D \log(N/D))$ lower bound for broadcast in radio networks. *SIAM Journal on Computing*, 27(3):702–712, 1998.

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

24 Prabhakar Raghavan and Eli Upfal. Stochastic contention resolution with short delays. *SIAM Journal on Computing*, 28(2):709–719, 1998.

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