Computer Communications 181 (2022) 58–68

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Making slotted ALOHA efficient and fair using reinforcement learning

Molly Zhang, Luca de Alfaro *, J.J. Garcia-Luna-Aceves

University of California Santa Cruz, United States of America

ARTICLE INFO

Keywords:
Channel access
ALOHA
Reinforcement learning

ABSTRACT

Reinforcement learning (RL) has been proposed as a technique that allows nodes to learn to coordinate their transmissions in order to attain much higher channel utilization. Several RL-based approaches have been proposed to improve the performance of slotted ALOHA; however, all these schemes have assumed that immediate feedback is available at the transmitters regarding the outcome of their transmissions. This paper introduces ALOHA-dQT, which is the first channel-access protocol based on the use of RL in the context of slotted ALOHA that takes into account the use of explicit acknowledgments from receivers to senders. As such, ALOHA-dQT is the first RL-based approach for channel access that is suitable for wireless networks that do not rely on centralized repeaters or base stations. ALOHA-dQT achieves high utilization by having nodes broadcast short summaries of the channel history as known to them along with their packets. Simulation results show that ALOHA-dQT leads to network utilization above 75%, with fair bandwidth allocation among nodes.

1. Introduction

Nodes simply transmit their packets without coordination in the ALOHA protocol, which makes it attractive because of its simplicity. However, the channel maximum utilization is limited to about 18%, and to about 37% if transmissions are organized in fixed-length time-slots. To improve the utilization of the channel, coordination among the nodes is essential, and a variety of medium-access control (MAC) protocols have evolved over the years to provide such coordination.

Recently, reinforcement learning has been proposed as a way to achieve inter-nodal coordination without the need for a central authority or complex signaling. The application of reinforcement learning (RL) to channel access has followed two main directions: using deep neural networks to learn general strategies [1], and using "expert-based" systems that learn which strategies to use among a fixed number of them [2–5]. The latter approaches are lighter-weight, and have been successful in achieving high channel utilization in networks with many active nodes.

The limitation of all the RL-based approaches proposed to date is that they assume that transmitters know the fate of their transmissions as soon as they are done transmitting. Emulating this in practice requires a central node using a secondary channel to either re-transmit what it receives from other nodes or transmit explicit acknowledgments to transmissions received without interference. Using implicit acknowledgments to drive the "reinforcement" in RL is an impediment for deployment and use in wireless networks based on distributed control in which such functionality cannot be provided.

This paper presents ALOHA-dQT, which is the first RL-based approach applied to slotted ALOHA that takes into account the use of explicit acknowledgments from receivers to senders. Accordingly, ALOHA-dQT is suitable for wireless networks with distributed control. The only requirement for the proposed scheme is a time-slotted channel. ALOHA-dQT is based upon ALOHA-QTF [4], and adds to it an explicit-acknowledgment scheme based on nodes transmitting their knowledge about channel history along with their packets. When nodes receive these channel histories from other nodes, they merge them into their own knowledge of history: these history updates drive the reinforcement learning. This history reconstruction is modeled on knowledge-monotonic, distributed computation in distributed systems [6,7].

Section 3 summarizes the common elements of ALOHA-QTF and ALOHA-dQT, and Section 4 presents the acknowledgment scheme of ALOHA-dQT, and how it is used to drive reinforcement learning. The reinforcement learning scheme of ALOHA-QTF is modified in ALOHA-dQT to account for the delay in learning about transmission outcome. The delay has an implication on which "experts" (strategies) are affected, but more deeply, the reinforcement needs to be modified. For instance, strategies that trigger transmissions need to be temporarily demoted until their transmissions are acknowledged, or else they might trigger "collision storms" under which no packet is received in periods of high contention.

ALOHA-dQT is well suited for network nodes that can detect the presence of radio energy during transmission slots, and thus can distinguish empty slots from slots where collisions occurred. However, it also works well with nodes without energy-detection capabilities.

Section 5 presents a performance comparison of ALOHA-dQT with previous protocols, including ALOHA-Q, ALOHA-QTF, and ALOHA with

* Corresponding author.
E-mail addresses: mollyzhang@ucsc.edu (M. Zhang), dealfaro@acm.org (L. de Alfaro), jj@soe.ucsc.edu (J.J. Garcia-Luna-Aceves).

https://doi.org/10.1016/j.comcom.2021.09.018
Received 31 March 2021; Received in revised form 2 July 2021; Accepted 20 September 2021
Available online 29 September 2021

exponential backoff. Simulation results show that ALOHA-dQT offers channel utilization that is generally above 75% if energy detection is available, and above 65% if it is not available, under a wide range of network dynamics. In addition to high utilization, ALOHA-dQT also achieves high fairness in sharing the bandwidth between the nodes, as measured using the Jain index. We also provide results on how the values of the protocol hyper-parameters affect the results in terms of both utilization and fairness.

2. Related work

The maximum channel utilization that can be achieved with the ALOHA protocol is about 18%. Slotted-ALOHA improves this to about 37% by confining transmissions to time slots with a length equal to a packet length [8]. To go beyond this limited channel utilization, some kind of coordination among nodes is needed. A popular approach used in the past to achieve inter-nodal coordination is via reservations, in which nodes declare their transmission needs, and a central authority assigns slots to individual nodes (e.g., [9]) or nodes engage in peer-to-peer signaling to establish reservations. Other schemes adopt repetition strategies with which each node transmits the same packet multiple times, and relying on physical-layer techniques (e.g., code division multiple access and successive interference cancellation) to improve throughput [10–13].

Reinforcement learning has been proposed as a technique to achieve coordination without requiring a central authority assigning slots, or mechanisms related to the physical channel. The idea is that nodes can observe the channel and learn how to coordinate their transmissions to reduce collisions and achieve high network utilization. The most powerful type of reinforcement learning is *deep reinforcement learning* (DRL), in which a neural net learns the success of actions (transmit, or wait) as a function of channel history [1]. Unfortunately, the very generality of the approach slows down the learning: adaptation has been demonstrated in [1] only for channels with two DRL nodes, and it takes tens of thousands of time slots even in such simple scenarios.

Expert-based learning is a less powerful type of reinforcement than DRL, but attains faster learning. Nodes in this approach learn which of different "experts", or transmission strategies, to follow [14–16]. In ALOHA-Q, nodes consider a transmission frame of fixed length L, and learns the quality q_i, $1 \leq i \leq L$, of each frame slot [2,3]. At each frame, ALOHA-Q transmits in the time-slot with highest q-value; if the transmission is successful, the q-value of the slot used is increased, and if the transmission is unsuccessful, the q-value is decreased. ALOHA-Q can reach high utilization when the frame length is well matched to the number of active nodes, even though adaptation is not always fast, a function of the particular kind of updates used.

The approach based on experts has also been used coupled with the *schedule trees* of [17]. In a schedule tree, each child schedule has twice the period of the parent, and sibling schedules transmit in different slots of the period. The variable period leads to *frameless* protocols that can adapt to different numbers of active nodes. In [17] the tree structure is used to resolve each collision as it arises. In ALOHA-QTF [4] and AT-ALOHA [5], each tree schedule is an "expert", and the nodes learn which experts to follow. In AT-ALOHA, each node tracks a subset of experts, and use a set of rules to update the set according to channel outcomes. In ALOHA-QTF, weights are associated with each expert, and nodes select schedules with high weights. ALOHA-QTF AND AT-ALOHA attain high network utilization under a wide range of network dynamics.

All of the channel-access approaches based on reinforcement learning mentioned above rely on implicit, immediate acknowledgments of transmissions. This type of feedback can be emulated with a centralized repeater that rebroadcasts all received packets on a separate orthogonal channel. The immediate feedback is used to update weights (ALOHA-Q and ALOHA-QTF) or update the experts in use (AT-ALOHA). We adopt the schedule-tree and schedule weights of ALOHA-QTF, and add to it an explicit-acknowledgment scheme based on nodes broadcasting, along with each packet, their knowledge of past channel history. Updates to the knowledge about channel history trigger weight updates. The history broadcasting and update is modeled after distributed monotonic computation in distributed systems [6,7].

Concurrent to the work on ALOHA-dQT, we developed a related approach to delayed acknowledgment in medium access with tree-based schedules [18]. While the performances of the approaches are comparable, the techniques used are different. The policy trees of [18] do not have quantitative weights associated with them; rather, a set of active nodes is kept, and the learning is driven by dynamically updating such set rather than learning the weight of each node in the tree. The acknowledgment mechanism is also different. Nodes executing ALOHA-dQT broadcast their knowledge about the recent channel history to build a consensus version of it. By contrast, the acknowledgments used in the approach presented in [18] are sent via a gossip protocol.

3. Learning the schedules

ALOHA-dQT, like its predecessor ALOHA-QTF [4], is a protocol for fully-connected networks in which the channel is time-slotted. At each time slot a node can either transmit (T) or wait (W), and the channel outcome can be either empty (E), if no node transmitted; success (S), if exactly one node transmitted, or collision (C), if two or more nodes transmit. Both ALOHA-QTF and ALOHA-dQT use reinforcement learning to allow the nodes to coordinate, and schedule their transmissions in a way that reduces collisions while allocating bandwidth fairly.

The reinforcement learning and node adaptation in ALOHA-QTF are driven by immediate feedback regarding each slot outcome in $\{E, S, C\}$, as soon as the transmission slot ends. This is not practical in single-channel wireless networks based on distributed control, because nodes in these networks must use their radios in either transmit or receive mode in each time-slot, and a sender can learn the outcome of its transmission only by receiving an acknowledgment from other nodes. ALOHA-dQT differs from ALOHA-QTF by the use of an acknowledgment mechanism and in how the reinforcement learning is driven. ALOHA-dQT drives the reinforcement learning with a mix of information gleaned from observing the network and information received via acknowledgments. However, the two protocols share the same policy structure and the same operations on such structure, which are covered in this section. The acknowledgment structure and how the information drives reinforcement learning. are presented in the following section.

3.1. The policy tree

In the protocols considered in this paper, as in ALOHA-QTF of [4], nodes transmit according to the union of periodic schedules. Each node keeps a local time-slot counter t; these counters need not be synchronized across the network. A *policy* consists in a (periodic) *schedule* $\sigma = (i, m)$, which prescribes transmitting at all times t such that $t \mod 2^m = i$; the schedule has period 2^m and offset i. For $\sigma = (i, m)$, we let $\delta(\sigma, t)$ be 1 if $t \mod 2^m = i$ and 0 otherwise, so that $\delta(\sigma, t)$ is the indicator function of the transmit times of σ.

A node uses as policies the set of schedules $\mathcal{P} = \{(i, 2^m) \mid 0 \leq i < 2^m, 0 \leq m \leq n\}$, up to some periodicity 2^n. For each policy σ in $\mathcal{P}$, the node stores a *weight* $w_\sigma \in [0, 1]$ representing the quality of the policy, that is, its ability to prescribe transmitting without causing collisions.

The policies can be organized into a tree, illustrated in Fig. 1, where policy (i, m) has as children $(i, m + 1)$ and $(i + 2^m, m + 1)$. The nodes at the same tree level have the same period but different offsets, and thus prescribe non-colliding transmissions; every child node transmits in half of the slots of the parent. To coordinate the transmissions of N nodes, if the nodes can be all active at the same time, a policy tree with at least N leaves is needed. In practice, a tree with depth at least $n = \lfloor \log_2 N \rfloor + 2$ is preferable, to allow nodes to modulate their bandwidth.

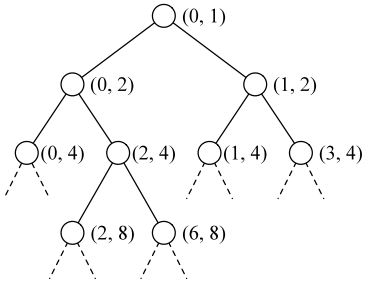

Fig. 1. Policy tree in ALOHA-QTF.

The periodic structure of the policies, and their hierarchical organization, facilitates the learning process of the nodes. In fact, a node of depth n contains $2^{n+1} - 1$ policies, yet every policy conflicts with only n others: thus, if we pick two policies at random, it is rare that they conflict (for $n = 8$, the probability is ≈ 0.016). Further, if two policies conflict, they are guaranteed to do so at least every 2^n time slots, facilitating conflict detection. These two properties, that conflicts are rare, and are detected early, are crucial in driving adaptation. We experimented using as policies the (larger) set of schedules "transmit when $t \mod k = i$" for $0 \le i < k \le 2^n$. In this set, conflicts are common, and can be often discovered only with delay, as two schedules with periods k_1, k_2 cause a collision only once every minimum common multiple of k_1, k_2. Using this larger set of schedules prevented nodes from adapting, and yielded very poor performance: more freedom of behavior did not translate in better adaptation.

Another fundamental property of the policy tree is that it is invariant with respect to clock offsets among nodes. Specifically, let $(\mathcal{P}, w)$ be a policy tree with its set of weights. If we translate time by Δ, so that the original time t and translated time t' are related by $t' = t + \Delta$. The weighed policy tree $(\mathcal{P}, w)$ for time t is equivalent to the weighed tree $(\mathcal{P}, w')$ for t', where $w'_{(i',m)} = w_{(i,m)}$ for $i' = (i + \Delta) \mod 2^m$, in the sense that the two weighed trees $(\mathcal{P}, w)$ and $(\mathcal{P}, w')$ prescribe the same actions at the corresponding. Further, the relation that relates (i, m) in $\mathcal{P}$ with $((i+\Delta) \mod 2^m, m)$ in $\mathcal{P}'$ preserves the tree structure, in the sense that related nodes have related children. This has the consequence that *clock synchronization among nodes is not required:* every node can keep its own local clock, which is incremented at each transmission slot; offsets between clocks at different nodes simply correspond to different arrangements of weights in the policy trees of the nodes.

3.2. Policy-tree protocols

Policy-tree protocols, such as the ALOHA-QTF protocol of [4] and the protocols presented in this paper, consist in a *weight initialization step*, which is then followed by three steps performed cyclically:

1. *Policy selection:* at the start of each network time-slot, a set of *active policies* is selected; these policies drive the decision to transmit, or to wait.
2. *Weight update:* at the conclusion of each network time-slot, the weights of all policies are updated according to the outcome of the time slot.
3. *Weight normalization:* following the weight update, the weights are renormalized: the weight from "losing" policies is redistributed to other policies, and the numerical values of the weights are renormalized to ensure that they fall within a prescribed range.

We describe these steps in detail below.

Weight initialization. The weight of policy $(i, m) \in \mathcal{P}$ is initialized by:

$$w_{(i,m)} = \beta \cdot \frac{0.9 + 0.1 \cdot X_{(i,m)}}{1.2^m} , \tag{1}$$

where $\{X_\sigma\}_{\sigma \in \mathcal{P}}$ is a set of random variables independently sampled from the uniform distribution over $[0, 1]$. In [4] the value $\beta = 0.2$ was used. The denominator 1.2^m makes it so that nodes are initially likely to adopt policies that transmit frequently, falling back on policies that transmit more rarely only as needed to avoid collisions. This allows a faster ramp-up of network utilization. Choosing completely random initial values would slow down somewhat the initial adaptation of the nodes. This initialization has a very small impact on performance, since it happens only when a node is powered up, rather than each time it has new packets to send.

Policy selection and transmission decision. At a time t, for a weight vector w for the policies, the set of *active* policies is

$$\mathcal{A}_t = \{\arg\max_\sigma w_\sigma\} \cup \{\sigma \mid w_\sigma \ge w_h\} , \tag{2}$$

where w_h is a predefined threshold. In words, the active policies include the best-performing policy, along with all policies with weight above a given threshold w_h. In our implementations, we use $w_h = 0.95$. Thus, more than one policy can be active, enabling nodes to utilize a flexible amount of bandwidth. A node transmits at time t if one of its active policies at time t prescribes transmission, or $\sum_{\sigma \in \mathcal{A}_t} \delta(\sigma, t) > 0$, and waits otherwise. The decisions to transmit or wait is indicate with T or W.

Weight update. Given a time t' (not necessarily equal to the current time), an update factor $\alpha > 0$, and an amount of randomness $\gamma > 0$, the multiplicative update of the weights w is performed by

$$w'_\sigma = w_\sigma \cdot \exp\left(\alpha\, X_\sigma^\gamma\, \delta(\sigma, t)\right) . \tag{3}$$

where $\{X_\sigma\}_{\sigma \in \mathcal{P}}$ is a set independent random variables sampled uniformly at random from the interval $[0, 1]$. This update is written simply as $w' = U(w, \alpha, t', \gamma)$. Thus, only the weights of the policies active at t' are updated, and the update is randomized, to help breaking ties between nodes that lay claim to the same transmission slot.

Weight normalization. After the multiplicative updates are performed, the weights are normalized in a two-step process.

First, some of the weights lost by policies that are downgraded is redistributed across all policies. This is a classical technique in expert-based reinforcement learning, which facilitates transitioning to new experts when previous experts (in this paper, policies) become less effective [14,15]. Let w_σ, w'_σ be the weights of policy σ before and after the multiplicative update step, and let $S = \sum_{\sigma \in \mathcal{P}} w_\sigma$ and $S' = \sum_{\sigma \in \mathcal{P}} w'_\sigma$. Let $\Delta = S - S'$ be the decrease in total weight. If $\Delta > 0$ and $W' < w_{init} \cdot |\mathcal{P}|$, where w_{init} is the initial reputation given to each policy, we redistribute the lost weight via:

$$w'_\sigma := w'_\sigma + \Delta \frac{X_\sigma}{\sum_\sigma X_\sigma} ,$$

where $\{X_\sigma\}_{\sigma \in \mathcal{P}}$ is a set of random variables independently sampled from the uniform distribution over $[0, 1]$. Thus, the redistribution of the lost weight is randomized, again to break the symmetry between the updates at different nodes.

Second, the weights of all policies is bound to the $[q_{floor}, 1]$ interval, setting:

$$w_\sigma = \max(q_{floor}, \min(1, w_\sigma)) . \tag{4}$$

The normalization operation is summarized by $w' := normalize(w, t)$.

3.3. The ALOHA-QTF protocol

The ALOHA-QTF protocol [4] is a policy-tree protocol. Its weight update is as follows. At each time slot t, a node makes a decision

$d \in \{W, T\}$ to transmit (T) or to wait (W), the outcome $h \in \{E, S, C\}$ of the time slot is available as soon as the time slot is concluded, where E indicates an empty slot, S indicates a successful transmission by some node, and C indicates a collision occurred. Once the outcome is received, the network performs a multiplicative weight update $w' = U(w, \alpha, t', 1)$, where α is given by:

$$\alpha = \begin{cases} 0.2 & \text{if } (d,h) \in \{(W,E),(T,S)\}; \\ -0.5 & \text{if } (d,h) \in \{(W,S),(W,C),(T,C)\}. \end{cases} \quad (5)$$

Thus, the weight is boosted when a slot is available for the node to use, and is decreased when other nodes are transmitting into the slot. The weights are then re-normalized via $w' := normalize(w, t)$ as described in Section 3.1, using a threshold $q_{floor} = 0$.

To this basic scheme are added two enhancements to ensure fairness (see [4] for the details of the implementation).

The node measures the *requested* bandwidth b_r and *fair* bandwidth b_f. The requested bandwidth b_r is the fraction of network slots the node is currently transmitting at. The fair bandwidth b_f is obtained as $b_f = 1/\max(1, \hat{N})$, where $\hat{N}$ is an estimate of the number of active nodes obtained by collecting the distinct sender IDs collected in the last 2^{n+1} time-slots. Once these bandwidths are available, the protocol implements two enhancements before the policy normalization step.

First, if a node is using more than its fair share of the bandwidth, or $b_r > b_f$, the node will set to zero the weight of its active policies with a small probability $\epsilon_r > 0$ at every step. This ensures that nodes that use more than their fair share eventually relinquish transmission slots, which are then captured by other nodes.

Second, the multiplicative update step (5) is performed not according to α, but according to α' given by:

$$\alpha' = \alpha \cdot \begin{cases} \min(1, (b_r/b_f)^{1/2})) & \text{if } \alpha < 0; \\ \max(0, 1 - (b_r/b_f)^2)) & \text{if } \alpha \geq 0. \end{cases}$$

This modified update makes it easier for nodes with less than their fair share of bandwidth to gain more transmission slots, and for nodes with more than their fair share of bandwidth to relinquish their slots. In ALOHA-dQT we adopt these fairness enhancements as well.

4. ALOHA-dQT

The ALOHA-dQT protocol is designed for networks in which an immediate-acknowledgment mechanism is not possible and transmitters must be informed of the outcome of their transmissions via acknowledgments. We thus introduce an acknowledgment mechanism, and we show how the information the nodes glean from it is used to drive adaptation and learning. We distinguish between two kind of receivers:

- *Energy-detecting* receivers can distinguish between empty slots, and slots in which a collision occurred, by measuring the amount of energy carried in the channel during a time-slot. When such receivers detect energy, but cannot decode any packet, a collision is inferred.
- *Non-energy-detecting* receivers are the simplest ones, and they can only tell whether in a time slot, a packet could or could not be decoded.

We present acknowledgment mechanisms and protocol adaptations that apply to both of these kinds of receivers, leading to our proposed protocol, ALOHA-dQT. While ALOHA-dQT can be used both for networks with energy-detecting and non-energy-detecting nodes, some of the numerical constants used for weight update and normalization have different optimal values for networks comprising different kinds of nodes. In Section 5 we will discuss in detail the changes in adaptation coefficients that best accommodate networks of nodes with, and without, energy detection.

4.1. Acknowledgments via channel histories

In ALOHA-dQT, every node stores a *channel history* of the last N time-slot outcomes (in our experiments, we use $N = 16$). This history represents the knowledge the node has regarding what occurred in the last N time slots. Whenever a node transmits a packet, it attaches to it its channel history. When a node receives a packet, it takes the channel history received with the packet, which represents the best reconstruction of what occurred as known to the *sender* node, and merges it into its own channel history. This *history revision* step merges the information in the two histories: for instance, if the current node stored a T (Transmit) for a time slot in the history, and the other node stored an s (successful reception), the current node can update the time-slot information in its history to S (successful transmission). This process of history revision is what drives the reinforcement learning: an update in a time slot in the history drives an update for the weights of the policies that were active in that time slot.

The process of history transmission and update can be also understood as a network-wide distributed monotonic reconstruction of the true history of the channel [6,7]. Each network node can see only one portion of the history, as it is deaf when transmitting. By constantly transmitting the version of the channel history known to them, and updating their history according to the transmissions by others, the nodes' stored histories will tend to converge to the true history of the network.

Channel histories. A channel history $\mathcal{H}$ consists of a sequence of N symbols $\mathcal{H} = [h_0, \ldots, h_{N-1}]$, where symbol h_i represents the channel at time $t - i$. We denote by $\mathcal{H}_i$ the symbol h_i in position i of the history. A channel history time-slot can contain one of the following symbols:

- $\perp$ (bottom). There is no information for the slot yet. This will be changed into T or W once the node decides to transmit or wait.
- T (transmission). The node has transmitted in the time slot, and the outcome is not known yet.
- W (wait). The node has not transmitted, and its radio was in receive mode. No packet was decoded, and it is not known yet whether the slot was truly empty or whether a collision occurred. This state is used in non-energy-detecting nodes only.
- E (empty). The node has not transmitted, and the slot is known to have been empty.
- C (own collision). The node transmitted into a slot, and there was a collision.
- c (other collision). The node did not transmit in the slot, but others did, and a collision ensued.
- S (own success). The node has transmitted in the slot, and the transmission was successful.
- s (other success). Another node has transmitted in the slot, and the transmission was successful.

We denote by H the set of all channel symbols. There are eight symbols, so that symbols can be encoded with three bits; a 16-slot history thus requires six bytes.

Channel history extension. At the completion of each time slot, a node first extends its history by adding a $\perp$ symbol for its most recent slot, and by discarding its now $N + 1$th slot. This $\perp$ symbol is then immediately replaced, as follows:

- If the node transmitted, $\perp$ is replaced by T.
- If the node decided to wait, and was in receive mode, the behavior differs according to whether the node can detect channel energy:
  - If the node can detect channel energy, $\perp$ is replaced by:
    - E if there was no energy,
    - c if there was energy but no packet was received, and
    - s if a packet was received.

Table 1
History merging: the merged value is indicated as function of the current and received values.

Current h	Received h'						
	T	W	S	s	C	c	E
T	C	C	C^a	S	C	C	C^a
W	c	W	s^a	s^a	c	c	E

[a]Cells should not occur under normal protocol conditions.

Table 2
An example of collision detection and successful transmission acknowledgment for nodes that do not detect slot energy. The boldface and over-line symbols track transmissions by n_1 and n_2, respectively.

t_1	$\mathcal{H}^1@n_1$				t_2	$\mathcal{H}^2@n_2$			
	$\mathcal{H}^1_3$	$\mathcal{H}^1_2$	$\mathcal{H}^1_1$	$\mathcal{H}^1_0$		$\mathcal{H}^2_3$	$\mathcal{H}^2_2$	$\mathcal{H}^2_1$	$\mathcal{H}^2_0$
6	W	s	W	$\mathbf{T}$	11	W	s	W	$\mathbf{W}$
7	s	W	$\mathbf{C}$	$\bar{s}$	12	s	W	$\mathbf{W}$	$\bar{T}$
8	W	$\mathbf{C}$	$\bar{s}$	T	13	W	$\mathbf{c}$	$\bar{S}$	s
9	$\mathbf{C}$	$\bar{s}$	S	W	14	$\mathbf{c}$	$\bar{S}$	s	W

– If the node cannot detect channel energy, $\perp$ is replaced by:

  * s if a packet was decoded,
  * W if nothing could be decoded.

Merging channel histories. Histories are merged using a function $r : H \times H \mapsto H$ that merges a symbol $h \in H$ with a received symbol $h' \in H$ into $r(h \triangleleft h') \in H$. To merge histories, we apply r element-wise, letting $\mathcal{H}''_i = r(\mathcal{H}_i, \mathcal{H}'_i)$ for all $0 \leq i < N$. The merging function is as follows.

- The state $\perp$ is the bottom knowledge state, and we have $r(\perp \triangleleft h) = h$ for all h.
- The states E, C, c, S, and s are *full* knowledge states, and are not updated, so $r(h \triangleleft h') = h$ for $h \in \{E, C, c, S, s\}$.
- The states T and W are *partial* knowledge states, and they are updated as in Table 1.

The rules in Table 1 can be understood as follows.

If the node transmitted ($h = T$), then a received s confirms reception, leading to S. All other received states, and in particular T, W, C, c, indicate that a collision occurred, either because some other node transmitted ($h' = T$), or because no packet could be decoded ($h' = W$), or because a collision was already determined to have occurred.

If $h = W$, no packet could be decoded, and the node, unable to detect channel energy, is unsure of the slot state. Since nothing could be decoded, any indication of transmission or collision ($h' = T, C, c$) indicates that a collision must have occurred. If $h' = E$, it means that another node was able through energy detection to determine that the slot was empty, and we accept that information.

Other combinations cannot occur under normal protocol conditions. In particular, a node cannot receive a notification that another node succeeded ($h' = S$) if the node transmitted ($h = T$) or did not receive ($h = W$), unless capture occurred. Similarly, a node that transmitted ($h = T$) cannot receive a report $h' = E$ of no energy in the time-slot, and a node that did not decode packets ($h = W$) cannot receive a report that someone else did decode a packet ($h = s$), unless capture occurred. For these combinations, Table 1 reports the safest conclusion the protocol can draw.

If we consider the information ordering $\{\perp\} < \{T, W\} < \{S, s, C, c, E\}$, where symbols in the same set are at the same level in the ordering, we see that the merging function r is monotonic in its first argument, so that $r(x \triangleleft y) \geq x$. Thus, the information each node has grows as acknowledgments are received, and the greater information is re-broadcast with the next packet. The nodes in a network are computing in distributed fashion a global information fixpoint.

Example: detecting collisions in networks that cannot detect channel energy. Table 2 illustrates how the acknowledgment mechanism enables a node to detect that a collision occurred, for nodes that cannot detect channel energy. We depict only the first 4 steps of history for two nodes n_1 and n_2; the nodes start at times $t_1 = 6$ and $t_2 = 11$ respectively (slot counters need not be the same across nodes); we depict the history at the end of each time slot.

- At step $t_1 = 6$, n_1 transmits and marks T in its history; node n_2 marks W, as a collision occurred and the node did not receive (nor it can detect the lack of energy).
- At $t_1 = 7$, n_1 receives a packet from n_2, and marks s in $\mathcal{H}^1_0$. It then updates $\mathcal{H}^1_1 := r(T \triangleleft \mathcal{H}^2_1) = r(T \triangleleft W) = C$, so that the W received from n_2 leads to update its transmission T into a C.
- At $t_1 = 8$, n_1 transmits a packet, which is received from n_2; n_2 marks s for the most recent history, and it updates the T for its own transmission into a S using $\mathcal{H}^2_1 = r(T \triangleleft \mathcal{H}^1_1) = r(T \triangleleft s) = S$.
- At $t_1 = 9$, the information about n_1's successful transmission is relied to n_1, so that $\mathcal{H}^1_1$ is set to S.

Packet retransmission. Packets are queued for retransmission when their transmission, initially labeled as T in the history, is updated to C, and they are considered as successfully transmitted when the history is updated to S. Furthermore, if a packet transmission labeled as T "slides out" of the fixed-length history still as T, the packet lacks acknowledgment, and it is also queued for re transmission.

4.2. Driving the learning

The history updates drive the updates to the weights of the policies. Initially, a position in the history contains the $\perp$ symbol; the position is then updated one or more times as a consequence of the outcome of the time slot, and of the subsequent history merging. When a position $0 \leq i < N$ is updated from h_i to h''_i at time t, we perform the multiplicative update

$$w' = U(w, \alpha_i, t - i, \gamma_i) \tag{6}$$

where the multiplicative coefficients α_i, and the randomization amounts γ_i, are specified in Table 3. In (6), the time $t - i$ is the absolute time to which history position i refers. The coefficients in Table 3 can be understood as follows.

- If the new state is T, we transmitted, but we have not yet received an acknowledgment (which would change the T into S). We deterministically demote the policies responsible for the transmission by a small amount until an acknowledgment is received. This ensures that during "collision storms" in which most outcomes are collisions and few acknowledgments are received, the nodes eventually back off from the policies that caused the collision storms.
- If the state is updated to S, we successfully transmitted, and we deterministically promote the policies that caused its use.
- If the state is updated to E, the slot is free, and we promote policies that make use of it using randomization to break ties among nodes claiming the slot.
- If the state is updated to C, c, or s, it means that there is contention in the use of the slot, and we demote policies that use the slot using randomization to break ties.
- Finally, if the state was W, and we receive a W, the state remains classified as W (last row of the table). If the node is non-energy-detecting, we promote slightly policies that would have made use of the time slot. We do this because non-energy-detecting nodes can never explicitly detect that a slot was empty (E): all they can do is, whenever other nodes report that there was no transmission in that slot (W), we increase the belief that the slot was empty, and we thus slightly promote policies that would have made use of the slot.

Table 3
Multiplicative update coefficients for (3) according to the update in channel outcome; h refers to the slot outcome before the update, and h'' to the slot outcome after the update.

Node type	h	h''	α	γ
All	$\perp$	T	−0.1	0
All	T	S	0.2	0
All	W	E	0.2	1
All	T, W	C, c, s	−0.8	1
Non-energy-detecting	W	W	0.01	1

The updates (6) are performed for all positions $1 \le i \le N$ of the history, after which the policy weights are re-normalized via $w' := normalize(w, t)$.

In transients periods when many nodes become active, many collisions may occur, to the point where no acknowledgments are received (the acknowledgments being also lost in the collisions). When this happens, the negative update coefficient α for the transition $h \to h'' : \perp \to T$ in the first row of Table 3 ensures that the nodes quench their sending rate. Further, active policies lose their weight the faster, the higher up they are in the policy tree: an active policy at level m causes transmissions, and loses weight, every 2^m slots, which is twice as fast than the weight loss rate of any of its children at level $m + 1$. Thus, in these congested transients, policies that are higher up in the tree will be abandoned sooner, in favor of policies lower down in the tree, which transmit less frequently. This implements a self-regulating back-off: when many collisions occur, nodes automatically shift to using policies that cause rarer transmissions. In Section 5 we report detailed results for the case in which 50 nodes turning on simultaneously. The results show that the node overcome the initial period of high collisions, when acknowledgments are lost, to achieve coordination.

From Table 3 and the above discussion, we see that the main difference between nodes that can, and cannot, detect slot energy lies in their ability to promptly react to empty channel slots. Energy-detecting nodes can immediately detect empty slots and promote policies that make use of them; we will see in Section 5 that they will be able to make use faster of bandwidth that becomes available. Non-energy-detecting nodes can detect empty slots only with some delay, and this will slow down somewhat their adaptation speed.

The ALOHA-dQT protocol is schematically presented as Algorithm 1. We note that the algorithm uses the same fairness improvements presented in Section 3.3.

Constants:
- $n = 8$: depth of policy tree;
- $N = 16$: history length;
- $\epsilon_r = 0.02$: probability of relinquishing a time-slot;
- $\beta = 0.3$: initialization value for (1);

State Variables:
- $\mathcal{P} = \{(i, m) \mid 0 \le i < 2^m, 0 \le k \le n\}$: policies;
- $\{w_\sigma\}_{\sigma \in \mathcal{P}}$: policy weights;
- $\mathcal{H}$: history;
- *active*: True if the node is active; false otherwise;
- $t \in \mathbb{N}$: time slot counter;
- $\hat{N}$: estimated number of active nodes;

Channel Variables:
- $d \in \{T, W\}$: decision (T : transmit; W : wait);
- $\lambda \in \{T, W, s, c, E\}$: channel outcome;

Initialization:
- $t := 0$;
- Initialize $\mathcal{H} = [\perp, \ldots, \perp]$, and initialize the policy weights using (1);

At every time slot:
- `// Decision`
- **if** $\sum_{\sigma \in \mathcal{A}_t} \delta(\sigma, t) > 0$ **then** $d := T$ **else** $d := W$;
- **if** $d = T$ **then** transmit a packet alongside $\mathcal{H}$;
- `// Reception`
- Listen for a packet, and receive channel outcome λ;
- **if** $\lambda = s$ **then** receive the packet and the history $\mathcal{H}'$;
- Shift the history: $\mathcal{H} := [\perp, \mathcal{H}_1, \ldots, \mathcal{H}_{N-1}]$;
- Let $\hat{N}$ be the number of different sender IDs seen in the last 2^{n+1} time slots;
- `// History update`
- $\mathcal{H}_0 := \lambda$;
- **if** $\lambda = s$ **then** $\mathcal{H} := r(\mathcal{H}, \mathcal{H}')$;
- `// Weight update`
- Perform the weight updates (6) corresponding to the history updates;
- `// Fair slot relinquishment`
- **if** $b_r > b_f$ **then with probability** ϵ_r **do** $w_\sigma := (1 - \delta(\sigma, t))w_\sigma$;
- `// Normalization and time increment`
- $w := normalize(w)$;
- $t := t + 1$;

Algorithm 1: ALOHA-dQT Algorithm.

4.3. Algorithm complexity

At each time slot, the amount of work done by Algorithm 1 is linear in the number of nodes of the policy tree, and thus, linear in the maximum number of active nodes in the network. The expensive operations in Algorithm 1 consist of vector operations, where the weights of all nodes, represented collectively as a vector of numbers, are updated. The vector nature of the expensive operations enables efficient implementation.

5. Performance evaluation

5.1. Protocols

We compare the performance of ALOHA-dQT with that of its direct predecessor, ALOHA-QTF [4], as well as with that of ALOHA-Q, the reinforcement-learning protocol proposed by [2,3], and ALOHA with exponential backoff, or ALOHA-EB. We note that all of these prior protocols, ALOHA-QTF, ALOHA-Q, and ALOHA-EB, rely on implicit, immediate acknowledgments, which gives them an advantage of ALOHA-dQT, which instead uses the mechanism of delayed acknowledgment based on transmission history merging and update.

We consider two types of networks with ALOHA-dQT nodes: networks in which nodes can detect energy (indicated in our results simply as ALOHA-dQT), and networks in which nodes cannot detect energy (indicated in our results as ALOHA-dQT-NE). We compare these two setups with ALOHA-QTF, described in [4] and summarized in Section 3, as well as ALOHA-Q and ALOHA with exponential backoff (ALOHA-EB).

ALOHA-Q. We implemented ALOHA-Q, the Q-learning version of slotted ALOHA proposed in [2,3]. Since in our simulations the number of active nodes is at most about 50, we use a frame length $L = 50$ for ALOHA-Q. We experimented with other values, and they yielded similar or worse performance.

ALOHA-EB. In slotted ALOHA with exponential back-off, which we denote as ALOHA-EB, every node has an initial transmission probability $p = 1/2$ when it becomes active. The node then updates the probability p whenever a collision, or an empty slot, is detected on the network, setting $p := q * p$ in case of collisions, and $p := \min(1, p/q)$ in case of empty slots, where q is a constant that determines adaptation speed; in our simulations we use $q = 0.9$. For large numbers of nodes, the

bandwidth utilization of ALOHA-EB reaches the optimal value of $1/e$, or about 37% [19].

5.2. Simulation setup

The simulations were written on top of a simulator we wrote in the Python programming language. The simulator is composed of two main components: a *network simulator,* and *node simulator modules*. The network simulator is quite simple: it takes the decisions of all nodes for every time slot, computes the outcome (empty, successful transmission, or collision), and relies the outcome to each node. The node modules implement each protocol algorithm at each node. For ALOHA-dQT, for instance, the node module implements the time-slot counter, the policy tree, and Algorithm 1. Protocol modules for ALOHA-EB, ALOHA-Q, and ALOHA-dQT-NE nodes can be similarly implemented.

5.3. Performance metrics

We evaluate protocols by measuring their network utilization and fairness, defined as follows.

Network utilization. A time slot can either be empty, or it can contain a successful transmission or a collision. We define the *network utilization* as the fraction of slots that contain successful transmissions. To compute the network utilization, we aggregate time slots in *blocks* of 100, and for each block we can compute the network utilization as the ratio of individual slots that contains a successful transmission. Using blocks of length 100 offers a compromise between having a fine time resolution, and computing meaningful statistics on each block.

Fairness. The *fairness* of a protocol indicates how equitably the bandwidth of the protocol is distributed among the nodes, and we use the *Jain's index* [20,21]. Assume that n nodes are active in a time block and let b_i be the number of successful transmissions in the slot by node $i \in [1, \ldots, n]$. Let $B = \sum_{i=1}^{n} b_i$ be the bandwidth in the slot. Jain's index is computed as $J = B^2 \left(n \sum_{i=1}^{n} b_i^2\right)^{-1}$.

Jain's index is a quantity between $1/n$ and 1; it is 1 for a perfectly fair distribution of the channel ($b_i = B/n$ for all i), and it is $1/n$ if only one node gets to use the channel.

5.4. Simulation scenarios

We compare the performance metrics of different protocols in three simulation scenarios: a *ramp* scenario, a *churn* scenario, and a *50-node* scenario. In the scenarios, we use a fully-connected single-channel time-slotted wireless network. We simulate each scenario 20 times, each time using a different seed for the random number generator; our figures report the average (as a line) and the standard deviation (as a shaded area) of the set of 20 runs.

Ramp scenario. In the ramp scenario, there are initially 10 active nodes. The number of active nodes then increases gradually to 50, with one node becoming active each time-block. Then, after 100 time-blocks, 30 nodes become inactive, one each time block, starting from the nodes that have been active the longest. The number of active nodes at each time block is summarized in Fig. 2(c).

Churn scenario. The churn scenario simulates the case of nodes becoming active or turning inactive at random. More specifically, we have 20 nodes in a network. Initially, only one of them is active. At every time block, every node has a probability 1/100 of switching state, from inactive to active or vice-versa. Thus, an average of one node per time block switches state. We ran the simulation for 200 time blocks. Fig. 3(c) shows the average number of active nodes throughout the simulation.

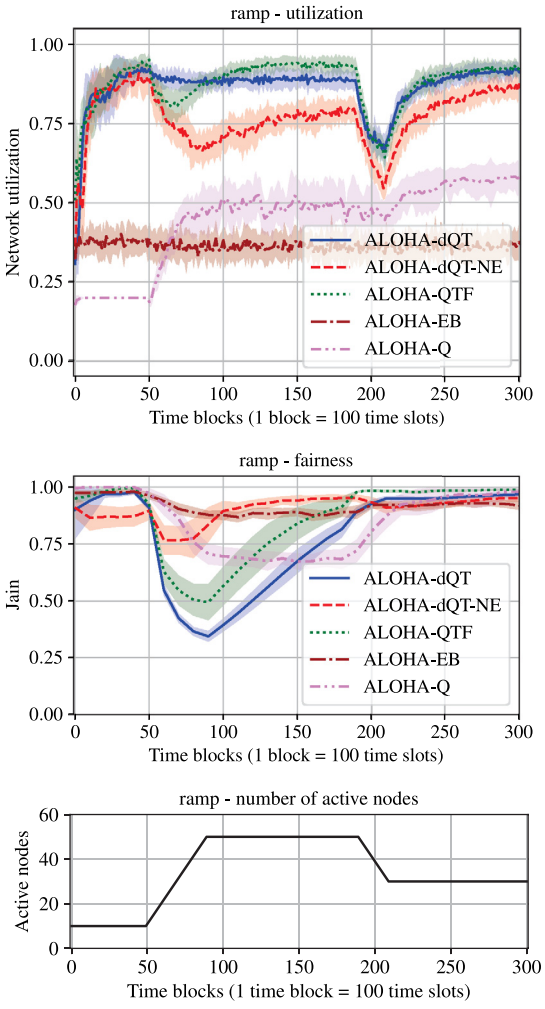

Fig. 2. Ramp experiment result. The solid lines are the average of 20 simulations; the colored bands are plus and minus one standard deviation.

50-node scenario. In the 50-node scenario, 50 nodes become active simultaneously after their (synchronized) turn-on. This scenario, while not particularly realistic, enables us to study how the protocol behaves during periods of such high contention that most transmissions result in collisions, and no acknowledgments are received.

5.5. Results

Comparison with ALOHA-QTF, ALOHA-EB and ALOHA-Q. The results for the ramp scenario are reported in Fig. 2, and those for the churn scenario in Fig. 3. We see from Figs. 2(a) and 3(a) that ALOHA-dQT and ALOHA-dQT-NE yield high network utilization, generally over 75%. If nodes can detect energy in network slots, as in the ALOHA-dQT setup, and thus differentiate empty slots from collisions slots, the performance is generally higher than in the ALOHA-dQT-NE setup, where energy cannot be detected. The performance of ALOHA-dQT approaches that of ALOHA-QTF, indicating that our delayed acknowledgments mechanism yields an efficiency which is almost as good as the ideal case of immediate acknowledgments. The performance for ALOHA-dQT-NE is slightly inferior to that of ALOHA-dQT, indicating that the ability to differentiate empty slots from collisions confers a clear, if relatively small, performance advantage.

For the ramp scenario, we see that after a brief transient, the network utilization for ALOHA-dQT is above 80% except in a brief transient when nodes become inactive, after about 200 time blocks.

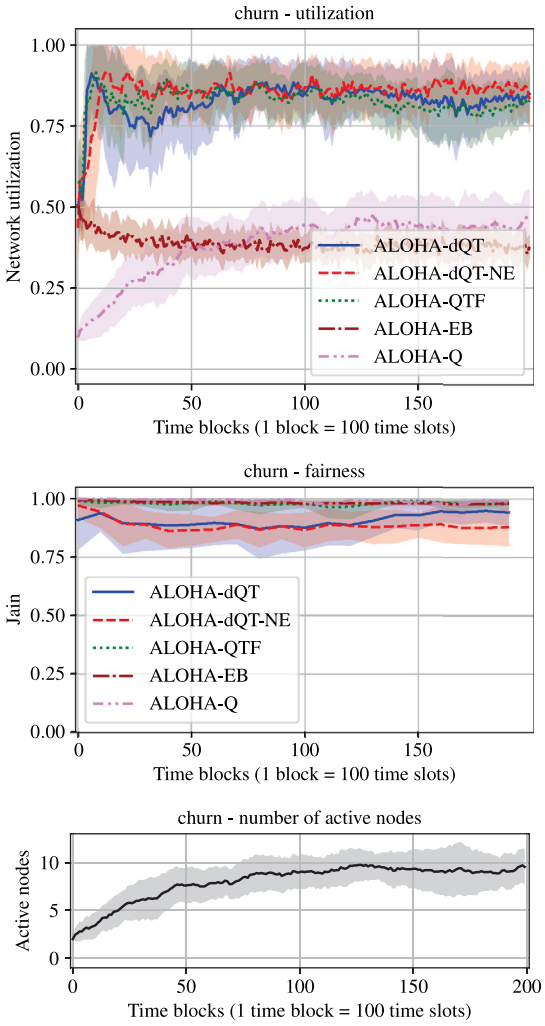

Fig. 3. Churn experiment result. The solid lines are the average of 20 simulations; the colored bands are plus and minus one standard deviation.

The utilization of ALOHA-dQT-NE is similar, but 10% to 15% lower. ALOHA-QTF has overall a slightly greater utilization than ALOHA-dQT. As for the other protocols, ALOHA-EB steadily tracks its optimal performance of 37%. ALOHA-Q does not offer optimal performance when the number of active nodes is 50, as one might expect. The reason is that when the number of active nodes is close to the frame length, even though the potential utilization is close to 1, the adaptation time is very long, on the order of hundred of thousands of time slots [2]. Instead, ALOHA-Q is able to reach better performance when the number of active nodes is 30. All the protocols exhibit acceptable fairness, except for a temporary dip when the number of active nodes is increasing. ALOHA-EB, due to its symmetry, offers superior fairness, if not superior utilization.

The utilization in the churn scenario follows a similar pattern, with ALOHA-QTF having highest utilization, closely followed by ALOHA-dQT, which at steady state offers utilization above 75%, and then by ALOHA-dQT-NE with utilization around 65%. ALOHA-EB is once again around 37%, and ALOHA-Q just below 50%. While in the ramp scenario the fairness of ALOHA-dQT-NE was slightly better than the one of ALOHA-dQT, the opposite is true in churn scenario.

In general, the fairness of ALOHA-dQT protocol can be improved at the cost of lower utilization, and vice versa. We can adjust both by using fairness parameter ϵ_r described in Section 3.3.

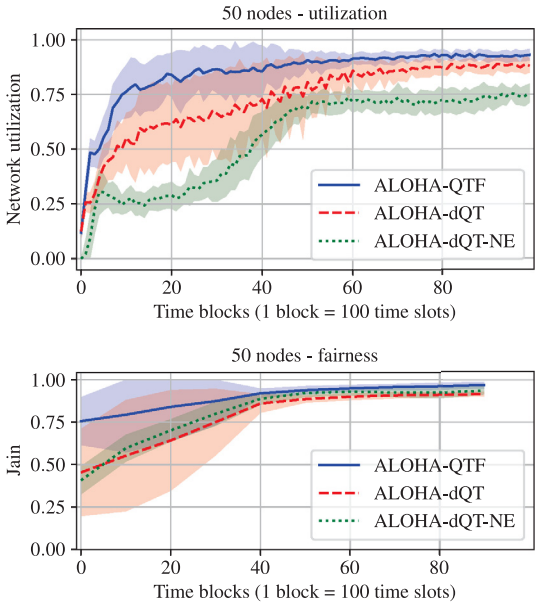

Fig. 4. 50-node experiment result. The solid lines are the average of 20 simulations; the colored bands are plus and minus one standard deviation.

Performance under simultaneous node startup. Fig. 4 reports the performance of ALOHA-QTF, ALOHA-dQT, and ALOHA-dQT-NE for the scenario in which 50 nodes become active simultaneously. In this case, we see that the network utilization ramps up slower in ALOHA-dQT than in its predecessor ALOHA-QTF. This is because ALOHA-dQT relies on explicit acknowledgments, and initially, there are so many collisions that most acknowledgments are lost. In this initial period, the adaptation of ALOHA-dQT is due to the fact that the weight of policies is decreased immediately after transmission (see the negative coefficient α in row 1 of Table 3), and nodes reduce policy weights also in response to collisions in which they do not participate (row 4 of Table 3). Thus, even if all acknowledgments are lost, the nodes can rapidly reduce their transmission rate and achieve coordination. Utilization reaches 50% in less than 1000 time-slots (or, so to say, 20 slots per node).

For ALOHA-dQT-NE, we see from Fig. 4 that it takes almost 4000 time-slots for utilization to reach 50%. The slower adaptation of ALOHA-dQT-NE is expected: the ability of ALOHA-dQT nodes to measure slot energy, and distinguish collisions from empty slots is particularly valuable in periods with very many collisions. In particular, in the initial period an ALOHA-dQT-NE node confuses the many collisions in which it does not participate with empty slots. The weight of the corresponding policies is slightly increased rather than decreased, slowing down adaptation (the last line of Table 3 applies). These results clarify the benefit of energy detection in periods of high congestion.

5.6. Hyper-parameter analysis

The performance of the ALOHA-dQT protocol depends on several parameters, including the choice of the update coefficients of Table 3, the relinquishment probability ϵ_r of ceasing transmissions in a slot, and the quality floor q_{floor} for the interval $[q_{floor}, 1]$ of qualities used. The update coefficients in Table 3 play a similar role in nodes with and without energy detection, and our analysis did not identify specific trade-offs or interesting variations in the choice of their values. On the other hand, the two latter quantities ϵ_r and q_{floor}, play a crucial role in determining protocol efficiency, fairness, and performance, and we offer here a more in-depth study of their influence on protocols with and without energy detection. The relinquishment probability ϵ_r is crucial in ensuring the fairness of the protocol, by ensuring that transmission

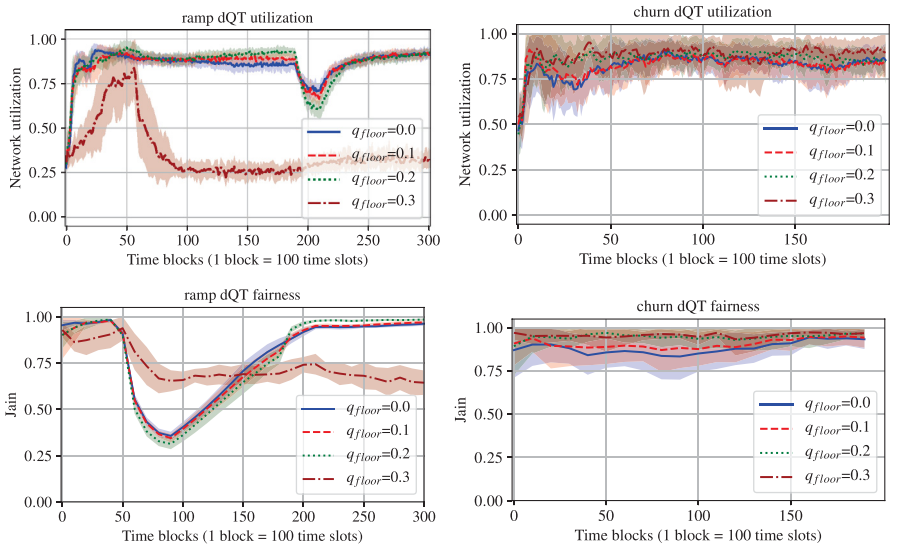

Fig. 5. Varying the quality floor q_{floor} in the ALOHA-dQT protocol.

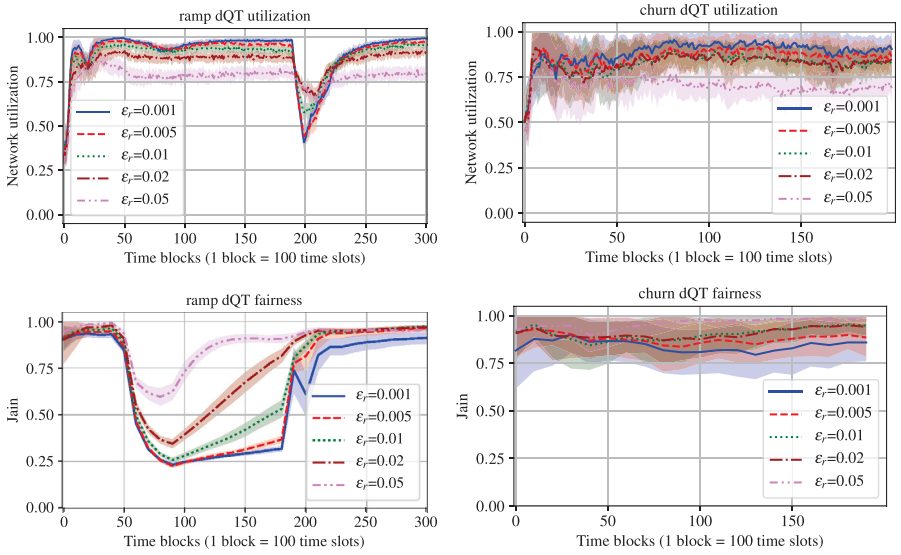

Fig. 6. Varying the relinquishment probability ϵ_r in the ALOHA-dQT protocol.

slots are not permanently held by the same nodes. Furthermore, for the reassignment of transmission slots to be effective, it is important that the policy quality be bounded away from 0. To see this, consider what happens at node B when a policy that was used in transmissions by node A is relinquished. At node B, several policies would have caused transmission in the slot: precisely, all policies (i, m) with $t \mod 2^m = i$. If the slot was in regular use by a periodic policy of node A, the policies of node B associated with the slot would have had a quality close to the quality floor q_{floor}, due to the negative quality updates occurring each time the slot is utilized by A. Thus, for node B to start utilizing the slot (or better, the periodic recurrences of the slot), it is necessary for the policy quality to climb from q_{floor} all the way to w_h (see (2)). If q_{floor} is very low, this takes a long time, leading to an ineffective recycling of slots. For the same reason, the choice of q_{floor} influences how fast nodes are able to start using slots that become available due to nodes stopping their activity: the higher q_{floor}, the faster empty slots are put back into service.

In general, the optimal values for these parameters depend on the node's ability to detect slot energy; we discuss the two cases independently.

ALOHA-dQT. In Fig. 5 we depict the effect of varying the quality floor q_{floor} in ALOHA-dQT, where all nodes can detect energy. Our base values for ALOHA-dQT are $q_{floor} = 0.1$, and $\epsilon_r = 0.02$. We see that values of q_{floor} greater than 0.2 can lead to markedly sub-optimal performance.

Fig. 6 gives the corresponding data for varying the slot relinquishment probability ϵ_r. We see that there is a trade-off between fairness, which is higher, the higher ϵ_r is, and utilization, which is higher when ϵ_r is lower — except in transition periods. Interestingly, in the transition periods of the ramp protocol, utilization benefits from higher fairness. This occurs because, when fairness is low, it is the original 20 nodes that monopolize a good share of the utilization, even when 50 nodes are active. When the original 20 nodes cease their activity, their departure causes a large temporary drop in utilization. The drop is less marked when fairness is higher, as under higher fairness these 20 nodes control a smaller share of the total utilization in the 50-node regime. Our choice for the relinquishment probability is $\epsilon_r = 0.02$.

ALOHA-dQT-NE. Figs. 7 and 8 report the corresponding analysis for the ALOHA-dQT-NE scenario, in which nodes cannot detect slot energy. Our chosen values are $q_{floor} = 0.3$ and $\epsilon_r = 0.005$. The interesting

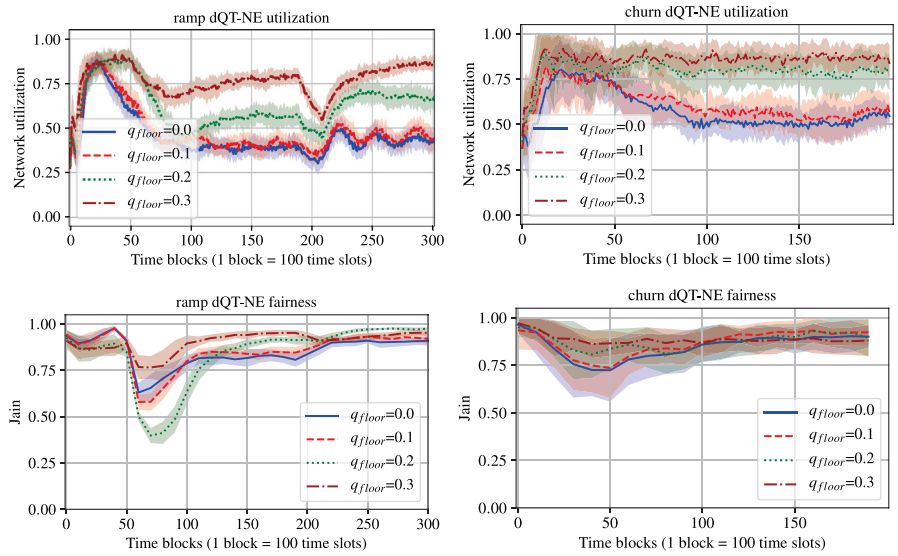

Fig. 7. Varying the quality floor q_{floor} in the ALOHA-dQT-NE protocol.

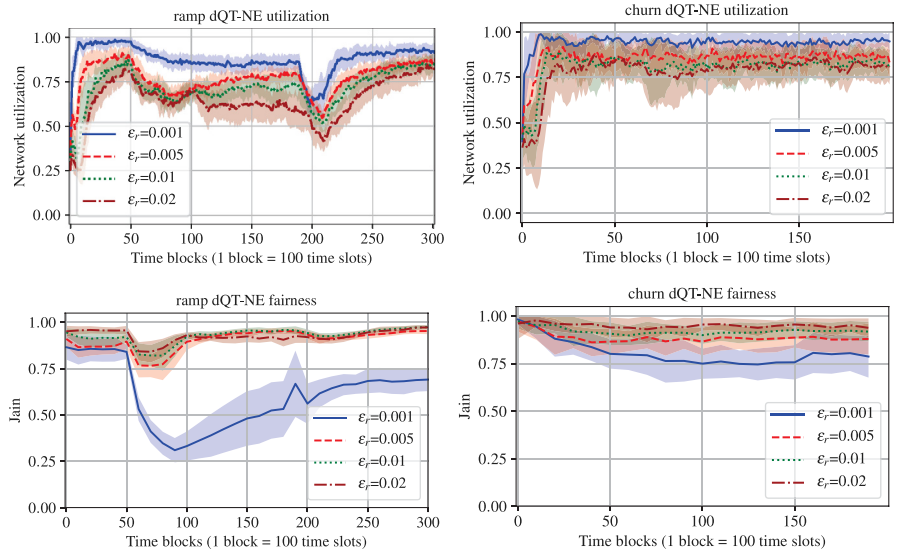

Fig. 8. Varying the relinquishment probability ϵ_r in the ALOHA-dQT-NE protocol.

result is that for ALOHA-dQT-NE, a higher quality floor q_{floor} is strongly beneficial, as indicated by Fig. 7. This is due to the fact that, in absence of energy detection, network nodes have a more difficult time distinguishing empty slots, and ramping up policy quality to exploit them. If qualities start from a higher floor, more empty slots end up being used, benefiting utilization. Further, as acquiring the use of empty slots is more difficult, our results indicate that it is beneficial to keep the relinquishment probability lower than in nodes where energy detection is possible.

6. Conclusions

We introduced ALOHA-dQT, a novel channel access protocol based on the use of reinforcement learning (RL) in the context of slotted ALOHA operating in a single-channel fully-connected wireless network. All previous variants of slotted ALOHA based on reinforcement learning, including ALOHA-Q [2,3], ALOHA-QTF [4], and the deep-RL approach of [1], assume that a transmitter knows the fate of its transmission at the conclusion of the time slot. In practice, this requires the presence of a repeater that rebroadcasts on a separate channel all packets or explicit acknowledgments. In contrast, ALOHA-dQT is based on explicit acknowledgments. The acknowledgment mechanism consists of nodes broadcasting and iteratively merging their information about the channel history. Updates to the information history drive the reinforcement learning and node adaptation. ALOHA-dQT offers high network utilization, generally above 75%, with fair allocation of bandwidth among active network nodes.

Channel access protocols based on RL hold the potential of offering high channel utilization, as the nodes can coordinate their behavior, and we view ALOHA-dQT as a first step in making these protocol suitable for practical use in wireless networks.

As mentioned in Section 5.6, the performance of the protocols we introduced depends on a set of hyper-parameters. In this paper, we have offered fixed values for these hyper-parameters, resulting in a particular trade-off between speed of adaptation, utilization, and fairness. In future work, it would be interesting to study whether it is possible to tune these hyper-parameters via reinforcement learning itself, carried out over longer periods of time, and driven by a goal function expressing the desired trade-off between utilization, speed of adaptation, and fairness. This would enable the protocol adaptation

itself to be tuned according to the particular characteristics and usage patterns of each network. For instance, in networks where nodes, once active, transmit for long periods of time (as in networks used for bulk data transfers), one can trade off adaptation speed for higher utilization. In networks where transmissions are short-lived, it may on the other hand be useful to prioritize fast achievement of fairness rather than utilization, to ensure that nodes can quickly receive their share of bandwidth when they become active.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Anyone at UC Santa Cruz (institutional conflict).

Acknowledgments

This material is based upon work sponsored by the Defense Advanced Research Projects Agency (DARPA), United States and the Air Force Research Laboratory (AFRL), United States. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA or AFRL.

References

[1] Y. Yu, T. Wang, S.C. Liew, Deep-reinforcement learning multiple access for heterogeneous wireless networks, IEEE J. Sel. Areas Commun. (2019).
[2] Y. Chu, P.D. Mitchell, D. Grace, ALOHA and q-learning based medium access control for wireless sensor networks, in: 2012 International Symposium on Wireless Communication Systems (ISWCS), IEEE, 2012, pp. 511–515.
[3] Y. Chu, S. Kosunalp, P.D. Mitchell, D. Grace, T. Clarke, Application of reinforcement learning to medium access control for wireless sensor networks, Eng. Appl. Artif. Intell. 46 (2015) 23–32.
[4] L. de Alfaro, M. Zhang, J. Garcia-Luna-Aceves, Approaching fair collision-free channel access with slotted ALOHA using collaborative policy-based reinforcement learning, in: IEEE IFIP Networking Conference, 2020.
[5] M. Zhang, L. de Alfaro, J. Garcia-Luna-Aceves, Collision-free channel access with delayed acknowledgements using collaborative policy-based reinforcement learning, in: ACM SIGCOMM Conference, NetAI Workshop, 2020.
[6] P. Alvaro, N. Conway, J.M. Hellerstein, W.R. Marczak, Consistency analysis in bloom: a CALM and collected approach, in: CIDR, 2011, pp. 249–260.
[7] N. Conway, W.R. Marczak, P. Alvaro, J.M. Hellerstein, D. Maier, Logic and lattices for distributed programming, in: Proceedings of the Third ACM Symposium on Cloud Computing, 2012, pp. 1–14.
[8] L.G. Roberts, ALOHA packet system with and without slots and capture, ACM SIGCOMM Comput. Commun. Rev. 5 (2) (1975) 28–42.
[9] G. Jakllari, M. Neufeld, R. Ramanathan, A framework for frameless TDMA using slot chains, in: 2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012), IEEE, Las Vegas, NV, USA, 2012, pp. 56–64, http://dx.doi.org/10.1109/MASS.2012.6502502, URL http://ieeexplore.ieee.org/document/6502502/.
[10] G. Liva, Graph-based analysis and optimization of contention resolution diversity slotted ALOHA, IEEE Trans. Commun. 59 (2) (2010) 477–487.
[11] E.E. Khaleghi, C. Adjih, A. Alloum, P. Mühlethaler, Near-far effect on coded slotted ALOHA, in: 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), IEEE, 2017, pp. 1–7.
[12] E. Paolini, G. Liva, M. Chiani, Coded slotted ALOHA: A graph-based method for uncoordinated multiple access, IEEE Trans. Inform. Theory 61 (12) (2015) 6815–6832.
[13] F. Schoute, Dynamic frame length ALOHA, IEEE Trans. Commun. 31 (4) (1983) 565–568.
[14] D.P. Helmbold, D.D. Long, B. Sherrod, A dynamic disk spin-down technique for mobile computing, in: Proceedings of the 2nd Annual International Conference on Mobile Computing and Networking, ACM, 1996, pp. 130–142.
[15] M. Herbster, M.K. Warmuth, Tracking the best expert, Mach. Learn. 32 (2) (1998) 151–178.
[16] O. Bousquet, M.K. Warmuth, Tracking a small set of experts by mixing past posteriors, J. Mach. Learn. Res. 3 (Nov) (2002) 363–396.
[17] J. Capetanakis, Generalized TDMA: The multi-accessing tree protocol, IEEE Trans. Commun. 27 (10) (1979) 1476–1484.
[18] M. Zhang, L. de Alfaro, M. Mosko, C. Funai, T. Upthegrove, B. Thapa, D. Javorsek, J. Garcia-Luna-Aceves, Adaptive policy tree algorithm to approach collision-free transmissions in slotted ALOHA, in: 2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), IEEE, 2020, pp. 138–146.
[19] L. Kleinrock, Queueing Systems. Volume I: Theory, wiley, New York, 1975.
[20] R.K. Jain, D.-M.W. Chiu, W.R. Hawe, A Quantitative Measure of Fairness and Discrimination, Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA, 1984.
[21] H. Shi, R.V. Prasad, E. Onur, I.G.M.M. Niemegeers, Fairness in wireless networks:Issues, measures and challenges, IEEE Commun. Surv. Tutor. 16 (1) (2014) 5–24, http://dx.doi.org/10.1109/SURV.2013.050113.00015, URL http://ieeexplore.ieee.org/document/6517050/.