Using Reinforcement Learning in Slotted Aloha for Ad-Hoc Networks

Molly Zhang
University of California Santa Cruz
mollyzhang@ucsc.edu

Luca de Alfaro
University of California Santa Cruz
luca@ucsc.edu

J.J. Garcia-Luna-Aceves
University of California Santa Cruz
jj@soe.ucsc.edu

ABSTRACT

Slotted ALOHA is known to have poor channel utilization (a maximum of 37% when average offered load is one packet per time slot). Reinforcement learning has recently been proposed as a technique that allows nodes to learn to coordinate their transmissions in order to attain much higher network utilization. All reinforcement-learning schemes proposed to date assume immediate feedback on the outcome of a packet transmission. We introduce ALOHA-dQT, a reinforcement-learning protocol that achieves high utilization by having nodes broadcast short summaries of the channel history as known to them along with their packets. Our simulation results show that ALOHA-dQT leads to network utilization above 75%, with fair bandwidth allocation among nodes. ALOHA-dQT is the first reinforcement-learning approach applied to slotted ALOHA suitable for ad-hoc networks without centralized repeaters.

CCS CONCEPTS

• **Networks** → **Network protocols**; *Network performance modeling*; • **Theory of computation** → **Reinforcement learning**.

KEYWORDS

Channel Access; MAC protocols; ALOHA; Reinforcement Learning

ACM Reference Format:
Molly Zhang, Luca de Alfaro, and J.J. Garcia-Luna-Aceves. 2020. Using Reinforcement Learning in Slotted Aloha for Ad-Hoc Networks. In *23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '20), November 16–20, 2020, Alicante, Spain.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3416010.3423231

1 INTRODUCTION

The appeal of the ALOHA protocol is its simplicity, as nodes can transmit their packets when needed without coordination. However, in networks with many active nodes, the channel maximum utilization is limited to about 18%, and to about 37% if transmissions are organized in fixed-length time-slots, which reduces the time periods during which transmitted packets can overlap and collide.

To improve the channel utilization, coordination among the nodes is essential, and a plethora 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 internodal 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 [19],and using "expert-based" systems that learn which strategies to use among a fixed number of them [4, 5, 7, 20]. 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 reinforcement-learning based approaches proposed to date is that they are based on immediate acknowledgements, which in practice requires a central node using a secondary channel to either retransmit what it receives from other nodes or transmit explicit acknowledgments to transmissions received without interference. The implicit acknowledgements are used to drive the "reinforcement" in RL are an impediment for ad-hoc networks, where such functionality cannot be provided.

In this paper we present an RL-based protocol, ALOHA-dQT, which is suitable for ad-hoc networks.The only requirement for the proposed scheme is a time-slotted channel. ALOHA-dQT is based upon ALOHA-QTF [7], and adds to it an explicit acknowledgement scheme based on nodes transmitting, along with their packets, their knowledge about channel history. Section 3 summarizes the common elements of ALOHA-QTF and ALOHA-dQT, and Section 4 presents the acknowledgement scheme of ALOHA-dQT, and how it is used to drive reinforcement learning.

When nodes receive histories from other nodes, they merge them into their own knowledge of history, and the history updates drive the reinforcement learning. The acknowledgement process is modeled on knowledge-monotonic, distributed computation in distributed systems [1, 6]. The driving of the reinforcement is modified, compared to previous protocols, to account for the delay with which outcome information becomes known. 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 suited both to network nodes that can detect the presence of radio energy during transmission slots, and thus can distinguish empty slots from slots where collisions occurred, as well as to nodes without such energy-detection capabilities.

Section 5 presents a performance comparison of ALOHA-dQT with previous protocols, including ALOHA-Q, ALOHA-QTF, and ALOHA with exponential backoff. Our 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. Secton 6 presents our conclusions.

MSWiM '20, November 16–20, 2020, Alicante, Spain

ACM ISBN 978-1-4503-8117-8/20/11.
https://doi.org/10.1145/3416010.3423231

2 RELATED WORK

In the original ALOHA protocol, the maximum utilization that can be achieved is about 18%. Slotted-ALOHA improves this to about 37% by confining transmissions to time slots with a length equal to a packet length [17]. To go beyond this limited channel utilization, some kind of coordination among nodes is needed. A popular approach used in the past to achieve internodal coordination is via reservations, in which nodes declare their transmission needs, and a central authority assigns slots to individual nodes (e.g., [12]) 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 [13, 15, 16, 18].

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 [19]. Unfortunately, the very generality of the approach slows down the learning: adaptation has been demonstrated in [19] only for channels with two DRL nodes, and it takes tens of thousands of time slots even in such simple scenarios.

A less powerful type of reinforcement learning that is much faster is *expert-based* learning, in which nodes learn which of different "experts", or transmission strategies, to follow [2, 8, 9]. In ALOHA-Q, nodes consider a transmission frame of fixed length L, and learns the quality q_i, $1 \le i \le L$, of each frame slot [4, 5]. 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 expert approach has also been used coupled with the *schedule trees* of [3]. 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 [3] the tree structure is used to resolve each collision as it arises. In ALOHA-QTF [7] and AT-ALOHA [20], 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 reinforcement-learning based approaches mentioned above rely on implicit, immediate acknowledgements 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 acknowledgement 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 [1, 6].

3 LEARNING THE SCHEDULES

ALOHA-dQT, like its predecessor ALOHA-QTF [7], 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 ad-hoc networks in which nodes 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 acknowledgement from other nodes. ALOHA-dQT differs from ALOHA-QTF by the use of an acknowledgement 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 acknowledgements. However, the two protocols share the same policy structure and the same operations on such structure, which are covered in this section. The acknowledgement structure and how the information drives reinforcement learning. are presented in the following section.

3.1 The Schedule Tree

In ALOHA-dQT, 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 (periodic) *schedule* $\sigma = (i, m)$ 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 the set of schedules $\mathcal{P} = \{(i, 2^m) \mid 0 \le i < 2^m, 0 \le m \le n\}$, up to some periodicity 2^n. For each schedule 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 Figure 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. The periodic structure of the schedules, and their hierarchical organization, facilitates the learning process of the nodes. In fact, a node of depth n contains

$2^{n+1} - 1$ schedules, yet every schedule conflicts with only n others: thus, if we pick two schedules at random, it is rare that they conflict (for $n = 8$, the probability is ≈ 0.016). Further, if two schedules conflict, they are guaranteed to do so every 2^n time slots. These two properties, that conflicts are rare, and are detected early, are crucial in driving adaptation.

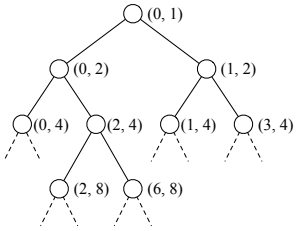

Figure 1: Policy tree in ALOHA-QTF

We experimented using 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.

Weight initialization. The weight of policy $(i, m) \in \mathcal{P}$ is initialized by: $w_{(i,m)} = \beta(1.2^{-m})(0.9 + 0.1 \cdot X_{(i,m)})$, where $\{X_\sigma\}_{\sigma\in\mathcal{P}}$ is a set of random variables independently sampled from the uniform distribution over $[0, 1]$. In [7] 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.

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

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

where w_h is a predefined threshold. In words, the active schedules include the best-performing schedule, along with all schedules with weight above a given threshold w_h. Thus, more than one schedule can be active, enabling nodes to utilize a flexible amount of bandwidth. A node transmits at time t if one of its active schedules 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

$$w'_\sigma = w_\sigma \cdot \exp\big(\alpha\, X_\sigma^\gamma\, \delta(\sigma, t)\big) . \tag{1}$$

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 schedules 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 schedules that are downgraded is redistributed across all schedules. This is a classical technique in expert-based reinforcement learning, which facilitates transitioning to new experts when previous experts (in this paper, schedules) become less effective [8, 9]. Let w_σ, w'_σ be the weights of schedule 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 $[0, 1]$ interval, setting $w_\sigma = \min(1, w_\sigma)$. The normalization operation is summarized by $w' := \mathit{normalize}(w, t)$.

3.2 The ALOHA-QTF Protocol

The ALOHA-QTF protocol [7] can be summarized as follows. The schedule weights are initialized as described in Section 3.1. 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 $w' = U(w, \alpha, t', 1)$ given in (1), where:

$$\alpha = \begin{cases} 0.2 & \text{if } (d, h) \in \{(W, E), (T, S)\}; \\ -0.5 & \text{if } (d, h) \in \{(W, S), (W, C), (T, C)\}. \end{cases} \tag{2}$$

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 $w' := \mathit{normalize}(w, t)$ as described in Section 3.1.

To this basic scheme are added two enhancements to ensure fairness (see [7] 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.

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 schedules 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 (2) 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-acknowledgement mechanism is not possible and transmitters must be informed of the outcome of their transmissions via acknowledgements. 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 protocol adaptations for both of these kinds of receivers.

4.1 Acknowledgements 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 schedules 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 [1, 6]. 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, by c if there was energy but no packet was received, and by s if a packet was received.
- If the node cannot detect channel energy, $\perp$ is replaced by s if a packet was decoded, and by 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.

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

Table 1: History merging: the merged value is indicated as function of the current and received values. Cells indicated with $*$ should not occur under normal protocol conditions.

	$\mathcal{H}^1@n_1$					$\mathcal{H}^2@n_2$			
t_1	$\mathcal{H}^1_3$	$\mathcal{H}^1_2$	$\mathcal{H}^1_1$	$\mathcal{H}^1_0$	t_2	$\mathcal{H}^2_3$	$\mathcal{H}^2_2$	$\mathcal{H}^2_1$	$\mathcal{H}^2_0$
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

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

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 acknowledgements 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.

Table 2 illustrates how the acknowledgement 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 acknowledgement, and it is also queued for retransmission.

4.2 Driving the Learning

The history updates drive the updates to the weights of the schedules. 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 $h''_i \neq h_i$ at time t, we perform the multiplicative update

$$w' = U(w, \alpha_i, t - i, \gamma_i) \tag{3}$$

where the coefficients α_i, γ_i depend on the new state h''_i, as specified in Table 3. In (3), 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 E, the slot is free, and we promote schedules that make use of it using randomization to break ties among nodes claiming the slot.
- If the new state is S, the slot is ours to use, and since the use has been successful, we deterministically promote the schedules that caused its use.
- If the new state is C, c, or s, it means that there is contention in the use of the slot, and we demote schedules that use the slot using randomization to break ties.
- Finally, if the new state is T, we transmitted, but we have not yet received an acknowledgement (which would change the T into S). We deterministically demote the schedules responsible for the transmission by a small amount until an acknowledgement is received. This ensures that during "collision storms" in which most outcomes are collisions and few acknowledgements are received, the nodes eventually back off from the schedules that caused the collision storms.

Updated h''	α	γ
S	0.2	0
E	0.2	1
C, c, s	-0.8	1
T	-0.1	0

Table 3: Multiplicative update coefficients as a function of updated channel outcome

In addition to these updates, a special update is included that is performed only by nodes that cannot detect energy. In these nodes, empty slots are never detected (unless another node that *can*

detect energy informs them that the slot was empty). To promote schedules that transmit into empty slot, we proceed as follows. If the current state of a slot is $h = W$, and the node receive $h' = W$, the outcome of the slot is not modified, as per Table 1. However, each time the node receives a W, it increases the likelihood that the slot was empty, for no other node so far has reported a collision or a transmission. To reflect this, if both $h_i = h'_i = W$, the following update is performed to give a small randomized incentive to using the slot in the future.

$$w' = U(w, 0.01, t - i, 1) \tag{4}$$

The updates (3) and (4) are performed for all positions $1 \leq i \leq N$ of the history, after which the policy weights are renormalized via $w' := normalize(w, t)$.

4.3 The ALOHA-dQT Protocol

The ALOHA-dQT protocol is schematically presented as Algorithm 1. We note that the algorithm uses the same fairness improvements presented in Section 3.2.

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 (??);

State Variables:
- $\mathcal{P} = \{(i, m) \mid 0 \leq i < 2^m, 0 \leq k \leq n\}$: schedules;
- $\{w_\sigma\}_{\sigma \in \mathcal{P}}$: schedule 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 schedule weights using (??);

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 (3) and (4) 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.

5 PERFORMANCE EVALUATION

We compare the performance of ALOHA-dQT, ALOHA-QT [7], ALOHA-Q [4, 5], and ALOHA with exponential backoff, or ALOHA-EB. We note that ALOHA-QT, ALOHA-Q, and ALOHA-EB rely on implicit, immediate acknowledgements, which gives them an advantage of ALOHA-dQT, which instead uses the mechanism of delayed acknowledgement 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 [7] and summarized in Section 3, as well as ALOHA-Q and ALOHA with exponential backoff (ALOHA-EB).

We implemented ALOHA-Q, the Q-learning version of slotted ALOHA proposed in [4, 5]. 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.

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% [14].

5.1 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* [10, 11]. Assume that n nodes are active in a time block and let b_i be the number of successful transmissions in

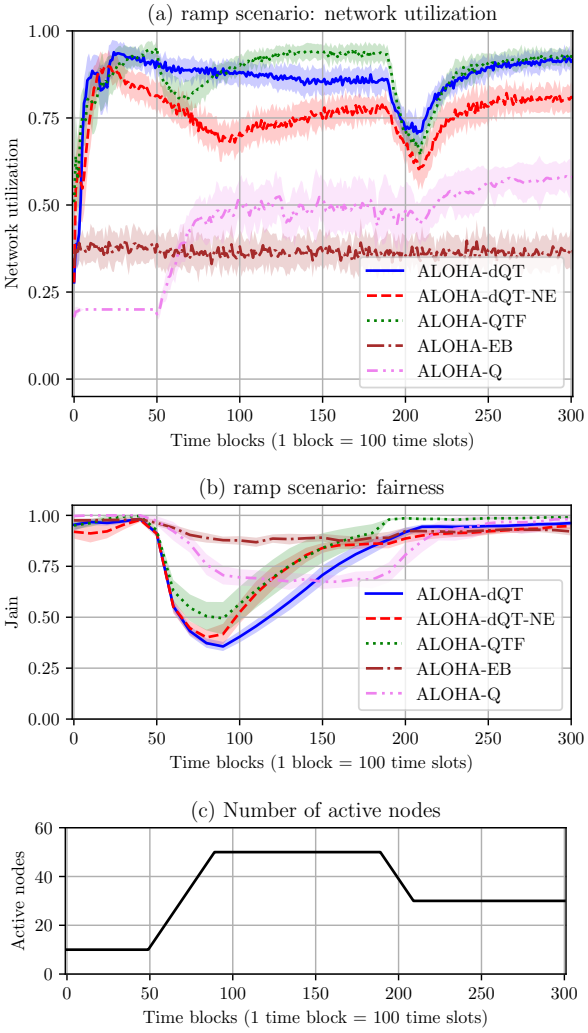

Figure 2: Ramp experiment result. The solid lines are the average of 20 simulations; the colored bands are plus and minus one standard deviation.

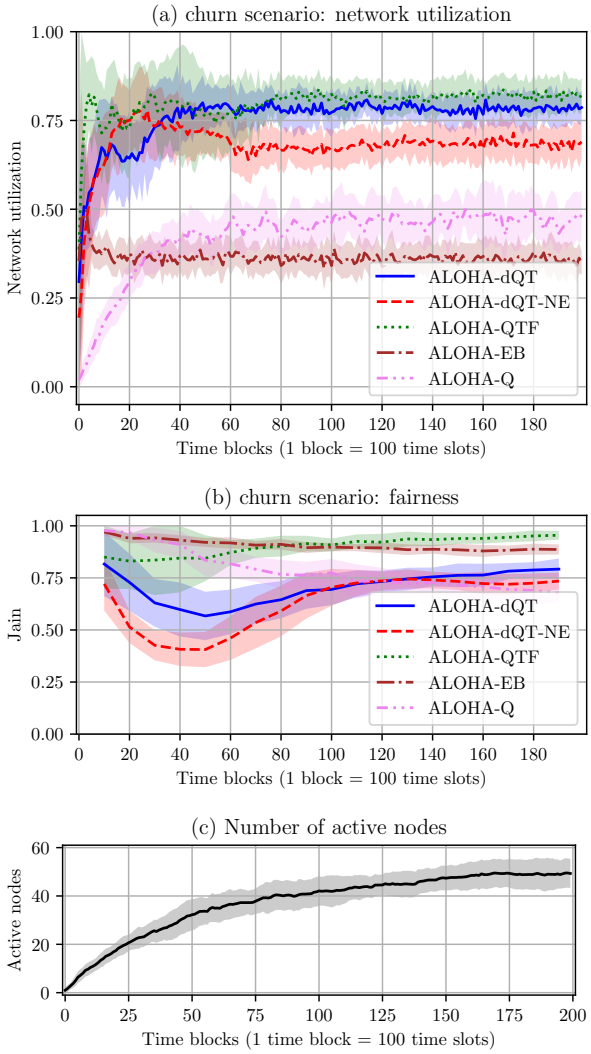

Figure 3: Churn experiment result. The solid lines are the average of 20 simulations; the colored bands are plus and minus one standard deviation.

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.

5.2 Simulation Scenarios

We compare the performance metrics of different protocols in two simulation scenarios: a *ramp* scenario and a *churn* scenario. In both 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 Figure 2(c).

Churn scenario. The churn scenario simulates the case of nodes becoming active or turning inactive at random. More specifically, we have 100 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. Figure 3(c) shows the average number of active nodes throughout the simulation.

5.3 Results

The results for the ramp scenario are reported in Figure 2, and those for the churn scenario in Figure 3. We see from Figures 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 acknowledgements mechanism yields an efficiency which is almost as good as the ideal case of immediate acknowledgements. 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.

In detail, 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. 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 procols, 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 [5]. 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 3.2.

6 CONCLUSIONS

We introduce ALOHA-dQT, a novel channel access protocol based on the use of reinforcement-learning 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 [4, 5], ALOHA-QTF [7], and the deep-RL based approach of [19], 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 an explicit acknowledgement protocol. The acknowledgement protocol is based on the nodes broadcasting, and iteratively merging, their information about the channel history, and 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.

Reinforcement-learning based channel access protocols 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 ad-hoc networks.

ACKNOWLEDGMENTS

This material is based upon work sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL). 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] P. Alvaro, N. Conway, J. M. Hellerstein, and W. R. Marczak. Consistency Analysis in Bloom: a CALM and Collected Approach. In *CIDR*, pages 249–260, 2011.
[2] O. Bousquet and M. K. Warmuth. Tracking a small set of experts by mixing past posteriors. *Journal of Machine Learning Research*, 3(Nov):363–396, 2002.
[3] J. Capetanakis. Generalized tdma: The multi-accessing tree protocol. *IEEE Transactions on Communications*, 27(10):1476–1484, 1979.
[4] Y. Chu, S. Kosunalp, P. D. Mitchell, D. Grace, and T. Clarke. Application of reinforcement learning to medium access control for wireless sensor networks. *Engineering Applications of Artificial Intelligence*, 46:23–32, 2015.
[5] Y. Chu, P. D. Mitchell, and D. Grace. ALOHA and q-learning based medium access control for wireless sensor networks. In *2012 International Symposium on Wireless Communication Systems (ISWCS)*, pages 511–515. IEEE, 2012.
[6] N. Conway, W. R. Marczak, P. Alvaro, J. M. Hellerstein, and D. Maier. Logic and lattices for distributed programming. In *Proceedings of the Third ACM Symposium on Cloud Computing*, pages 1–14, 2012.
[7] L. de Alfaro, M. Zhang, and 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.
[8] D. P. Helmbold, D. D. Long, and B. Sherrod. A dynamic disk spin-down technique for mobile computing. In *Proceedings of the 2nd annual international conference on Mobile computing and networking*, pages 130–142. ACM, 1996.
[9] M. Herbster and M. K. Warmuth. Tracking the best expert. *Machine learning*, 32(2):151–178, 1998.
[10] Huaizhou Shi, R. V. Prasad, E. Onur, and I. G. M. M. Niemegeers. Fairness in Wireless Networks:Issues, Measures and Challenges. *IEEE Communications Surveys & Tutorials*, 16(1):5–24, 2014.
[11] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 1984.
[12] G. Jakllari, M. Neufeld, and 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)*, pages 56–64, Las Vegas, NV, USA, Oct. 2012. IEEE.
[13] E. E. Khaleghi, C. Adjih, A. Alloum, and P. Mühlethaler. Near-far effect on coded slotted aloha. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–7. IEEE, 2017.
[14] L. Kleinrock. *Queueing systems. Volume I: theory.* wiley New York, 1975.
[15] G. Liva. Graph-based analysis and optimization of contention resolution diversity slotted aloha. *IEEE Transactions on Communications*, 59(2):477–487, 2010.
[16] E. Paolini, G. Liva, and M. Chiani. Coded slotted aloha: A graph-based method for uncoordinated multiple access. *IEEE Transactions on Information Theory*, 61(12):6815–6832, 2015.
[17] L. G. Roberts. ALOHA packet system with and without slots and capture. *ACM SIGCOMM Computer Communication Review*, 5(2):28–42, 1975.
[18] F. Schoute. Dynamic frame length aloha. *IEEE Transactions on communications*, 31(4):565–568, 1983.
[19] Y. Yu, T. Wang, and S. C. Liew. Deep-reinforcement learning multiple access for heterogeneous wireless networks. *IEEE Journal on Selected Areas in Communications*, 2019.
[20] M. Zhang, L. de Alfaro, and J. Garcia-Luna-Aceves. Collision-free channel access with delayed acknowledgements using collaborative policy-based reinforcement learning. In *ACM SIGCOMM Conference, NetAI Workshop*, 2020.