UNIVERSITY OF CALIFORNIA
SANTA CRUZ

POLICY TREE BASED REINFORCEMENT LEARNING APPROACHES FOR MEDIUM ACCESS CONTROL

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Molly Can Zhang

June 2021

The Dissertation of Molly Can Zhang
is approved:

Prof. Luca de Alfaro, Chair

Prof. J. J. Garcia-Luna-Aceves

Prof. David Helmbold

Prof. Jim Whitehead

Quentin Williams
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Policy Tree Based Reinforcement Learning Approaches for Medium Access Control

by

Molly Can Zhang

This thesis explores reinforcement learning (RL) based approaches to create adaptive Medium Access Control (MAC) protocols that learn from past transmission history. As apposed to canonical RL algorithms, a policy tree is used to represent both the decision space and the environment, by organizing potential transmission schedules in a binary tree. The protocols determine transmission schedule according to the policy tree, and also learns from the transmission outcome to update the policy tree, with the goal of maximizing both channel utilization as well as fairness of channel utilization. The updates are either editing the tree structure, or changing the weight of tree nodes, and these two mechanisms result in two set of algorithms: Adaptive Tree ALOHA and Quantitative Tree ALOHA. Both immediate and delayed acknowledgements mechanisms are created for both set of algorithms, begetting four families of policy tree protocols. This allows these protocols to be used in centralized wireless network as well as decentralized peer-to-peer ad-hoc networks. Policy tree based protocols outperform alternative MAC protocols, such as ALOHA with exponential backoff, ALOHA-Q and deep RL approaches, in terms of higher network utilization, faster learning time and high level of fairness in network bandwidth distribution.

Acknowledgments

I am very grateful for my thesis advisor, Luca de Alfaro. During the years that I studied with him, I am always humbled by his intelligence, knowledge, work ethic and insights, all delivered with friendliness and excellent sense of humor. I am grateful for the encouragement and support that he generously provided me. On top of teaching me how to be a computer scientist, he also inspired me to be a better person.

I also owe a great deal to J.J. Garcia-Luna-Aceves, who provides his insight, expertise and guidance in networking research, as well as a sense of lightheartedness and fun in all of our collaborations. I am also thankful for the funding from DARPA that supported part of work in this thesis. I also want to other members of my reading committee, David Helmbold and Jim Whitehead, for the valuable time, insight and guidance that they have committed to my thesis. I want to thank my graduate advisor, Alica Haley, who is always available and supportive with administrative obstacles.

I am grateful for the friendship and support of Rakshit, Johanna, Louisa, Yulia, Ross, Jay and Brigit. Their empathy, kindness and insights helped me persevere through various challenges, and their company made everything more meaningful.

I am really glad to have studied in UC Santa Cruz, the program empowers me to transition to a new career, all in a breathtakingly beautiful landscape. I am particularly fond of the redwood forest behind E2, which was a calming refuge in challenging times, and it also invigorated me with energy and life. Santa Cruz has successfully turned me into a much hippie-r person than I ever imaged, and I grateful for that.

Chapter 1

Introduction

In many networks, nodes communicate by sending and receiving messages over a shared communication medium, such as a frequency band or, in the old times, a coaxial cable [1, 28]. The medium is such that only one network node can transmit at any given time, otherwise collision occurs, which is a destructive interference that renders message unreadable. To avoid such destructive interference, and allow for the efficient transmission of information, *Medium Access Control* (MAC) protocols are employed to help coordinate the transmissions of the nodes. Each node in a network follows rules specified by the protocol to transmit data in order to avoid collision and to maximize the speed at which the data packets are transmitted.

Historically, there have been both schedule-based and contention-based protocols, such as TDMA and ALOHA, respectively. In schedule-based protocols, time is divided into fixed-length time slots, which are then assigned to the nodes by a central authority. The nodes then transmit according to this fixed schedule which they

are assigned. The collisions are avoided through central planning. In contention-based protocols, nodes transmit at any time, and re-transmit if they detect a collision. There are many variants of ALOHA protocols. The original protocol was developed at University of Hawaii, and was a radio protocol implemented with the help of repeaters, one repeater on each island [1]. Nodes would transmit as soon as they had a packet to transmit. If they detected the repeater re-transmitting their packet on a separate frequency band, then the packet is assumed successfully transmitted. If they do not detect a repeater re-transmission, the packet is assumed lost to a collision or other interference, and it is re-transmitted after a back-off delay that increases with each failed transmission attempt. The original ALOHA protocol was then extended in several directions. In *carrier-sensing* versions of the protocol, nodes transmit only when they do not detect transmissions; this led to the *Carrier-Sensing Multiple Access* (CSMA) protocols. In *slotted ALOHA* protocols, time is divided into discrete time slots, and nodes employ an ALOHA-type protocol, but beginning transmissions only at the beginning of each time slots; this reduces the time interval where collisions can arise to the beginning of each time slot, thus doubling the network throughput.

In these access protocols, the coordination between the nodes is either fixed (as in TDMA), or is limited to a backoff strategy. When nodes transmit according to a fixed schedule, network capacity is wasted whenever a node is allocated a transmission slot, yet has no information to send. When coordination is imperfect, and is limited to a backoff strategy, much network capacity is wasted in empty slots and collisions. In particular, lacking better coordination, ALOHA-type protocols can achieve utilization

limited to about 38% of the channel capacity.

In this thesis, we will explore the use of reinforcement learning to achieve node coordination and high channel capacity for ALOHA-type networks. We will use reinforcement learning to allow nodes to learn from each other's behavior, and coordinate dynamically their transmissions so as to avoid collisions. This will reap the low-collision benefits of fixed coordination schedules, while allowing for a flexible allocation of network capacity to the nodes. Our results will show that network utilization can reach about 80-90%, with a fair allocation to the nodes that are active (that have packets to be sent); furthermore, the network utilization remains high, and the allocation fair, even when the transmission loads of the nodes vary considerably in short time spans. That is, our protocols will be efficient, fair, and quick to adapt.

There are many approaches to reinforcement learning. Currently, *deep reinforcement learning* (DRL) is attracting much attention in the research community, due to its ability to learn complex behaviors from examples and row information. In perhaps the most striking demonstration of its capabilities, DRL has been shown to be able to learn how to play arcade games, on the basis of visual input (the CRT screen output) only [29], and to outperform Go game masters in the game of Go without human knowledge [37, 38]. DRL has been applied to the problem we are studying, of node coordination in MAC protocols. Unfortunately, while DRL is very powerful, it also requires much training data. In the context of a network, this means that the learning has to go on for quite a while, and this is a problem in two ways: first, network conditions can change during training, making what has been learned rather less useful; second,

the length of the training makes the protocol less efficient.

Here, we use another approach to reinforcement learning, called *expert-based learning* [17, 18, 6]. In this approach, a number of experts (each consisting in an algorithm) offer advice on when to transmit; the node learns which expert(s)' advice leads to successful transmissions, and follows those. This line of work has bee pursued already in the ALOHA-Q protocols of [10], where expert $1 \leq i \leq N$ of N experts was advising transmitting into the i-th slot of a group of N slots. ALOHA-Q still learn slowly, due to two factors. First, the N slots are not organized hierarchically in any way: thus, to win the use of k slots total, a node has to individually win the use of each of k slots, by transmitting in those and discouraging thus other nodes from using them. This battle for individual time slots is very inefficient. Second, ALOHA-Q used Q learning for weight update, which is sub-optimal compared to the updates to be introduced in this thesis later.

In this thesis, we propose to guide the search for compatible schedules via a hierarchical approach, via a *policy tree*. In our approach, as in ALOHA-Q, time is divided in time slots. Policy trees organize the policies according to their transmission frequency. The root of the tree is the (unrealistic) policy that transmits every time slot. The children of a given policy in the tree transmit half as much of the parent policy, alternating the time slots where they transmit. The policy tree has two properties that facilitate coordination. First, only policies where one is a descendant of the other collide: thus, if one picks two policies in the tree at random, the probability that they collide is low. Second, in our learning schemes, network nodes will tend to adopt top-level

policies in the tree, going down to lower level policies only when necessary. This will markedly facilitate agreement on efficient policies.

Protocols must not only be efficient, but also fair to the participating nodes, ensuring that the nodes that are active receive similar amounts of network capacity. We ensure fairness in the protocols by introducing parameters to tune the reinforcement learning. For instance, the level of "aggressiveness" and "politeness" of a node in claiming and relinquishing network capacity is modified by the amount of network capacity they are already using: a node will behave more "aggressively" if it is using less capacity and behave more "politely" if it is using more. This results in a long-term fair redistribution of network capacity.

We present four protocol families in this thesis [46, 48, 12, 47]. All four protocol families lead to 1). very high network utilization, generally over 80%, 2) fair bandwidth distribution, and 3) quick adaptation time.

The first two families are suited to networks with implicit acknowledgements, that is, networks where repeaters (as in the original ALOHA) let nodes know immediately about the success/failure of their transmissions. These two protocols, AT-ALOHA (for *Adaptive Tree* ALOHA) and ALOHA-QT (for *ALOHA Quantitative-Tree*), differ in the way they maintain the tree. ALOHA-QT, as the name Quantitative Tree suggests, associates with each policy in the tree a continuous weight value; at every round, it updates these values and transmit according to the policies with the highest weights. AT-ALOHA, on the other hand, just remembers the set of active policies, which constitute a subset of the tree structure, and uses a rule-based process for updating that

according to the result of network interactions.

The next two families of protocols, APT-ALOHA (for *Adaptive Policy Tree* ALOHA) and ALOHA-dQT (for *delayed ack Qualitative Tree* ALOHA), extend AT-ALOHA and ALOHA-QT to the case where explicit acknowledgements are needed. This means that a node hears back from its peer nodes the outcome of its transmission, usually a few time slots later. Delayed Ack mechanism allows for establishing consensus without needing a central planning or relaying device, and is thus more versatile. These protocols use two different approaches to acknowledgements: APT-ALOHA uses a gossip protocol, while ALOHA-dQT builds consensus through a shared recent history of transmissions. It is worth noting that APT-ALOHA and ALOHA-dQT are the first RL-based protocols that allow for delayed acknowledgements while improving on the performance of protocols relying on immediate acknowledgements.

In chapter 2, we review previous related work, in chapters 3-6, we present these four families of protocols - AT-ALOHA [46], ALOHA-QT [12], APT-ALOHA [48], and ALOHA-dQT [47], one in each chapter, and in chapter 7, we present a summary and comparison of the four families of protocols, and final conclusions.

Chapter 2

Previous Work

Several variants of ALOHA [1] have evolved over the years to allow more efficient sharing of common channels in wireless networks. The maximum channel utilization that can be achieved with the ALOHA protocol is about 18%. Slotted ALOHA [35] forces transmissions to occur at the beginning of time slots defined at the physical layer. This reduces the time during which transmissions are vulnerable to multiple-access interference (MAI) by half and hence doubles the maximum throughput attainable with pure ALOHA. A subsequent improvement on ALOHA was framed slotted ALOHA [31], which organizes time slots into transmission frames consisting of a fixed number of time slots; each node selects which time-slots to use in the frame. A few more schemes based on framed slotted ALOHA consist of using 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 [25, 23, 32, 36]. Jeong and Jeon [22] presented ALOHA with exponential

backoff (ALOHA-EB), where a node transmits in slot t with probability $p(t)$; this probability is updated via $p(t+1) = p(t)$ on success, $p(t+1) = p(t) \cdot q$ on failure, and $p(t+1) = p(t)/q$ on idle, where q is between 0 and 1.

The ALOHA protocol and its time-slotted variants achieve bandwidth sharing by transmitting greedily, and then adopting a backoff policy in case of collisions. This, however, leads to poor channel utilization. In ALOHA with exponential backoffs (EB-ALOHA), the bandwidth of network is bounded by $1/e \approx 0.37$ as the number of nodes grows, where e is the base of the natural logarithms. To address this limitation, a deeper level of coordination is needed in which nodes adapt to each other's behavior so that most transmission slots can be utilized without collisions or only a few.

Many schedule-based MAC protocols have been proposed in the past in which distributed algorithms are used assuming transmission frames consisting of a fixed number of time slots such that nodes select time slots in a way that eliminates multiple-access interference. The algorithms that have been proposed in this context include distributed elections of time slots for broadcast or unicast transmissions [4, 5, 26, 33], and the reservation of time slots based on voting and signaling similar to collision avoidance handshakes [44, 40, 41]. Some approaches allow the use of variable-length transmission frames by using lexicographic ordering of the identifiers of transmitting nodes, geographical or virtual coordinates related to the connectivity of nodes [14, 15, 26]. 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., [21]) or nodes engage in peer-to-peer signaling to establish

reservations.

The disadvantage of all these approaches is the added signaling complexity required to establish internodal coordination in order to attain TDMA schedules in a distributed manner. An alternative to achieving the desired coordination without complex signaling is to apply reinforcement learning (RL).

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 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. Deep reinforcement learning (DRL) has been used in other approaches to channel access in networks. In [30], DRL is used to choose which of N orthogonal channels to access using a MAC protocol, and in [42], it is used to choose a frequency channel in presence of interference. In [8], DRL is used for channel selection and access in LTE-U networks.

As a MAC protocol, the idea is that nodes can observe the channel and learn how to coordinate their transmissions to reduce collisions and achieve high network utilization. Approach by [45] is such an example. In particular, the approach relies on ResNet [16], a type of neural network that has skip connection across different layers. In this way, the value of an action can depend on the recent network history, and nodes can potentially learn arbitrary transmission schedules. The difficulty in applying deep reinforcement learning like this to medium access protocols is that the state-space can be very large. In a network with n nodes, the history of the past m transmission slots

gives rise to a state space of size at least n^m; large values of m are needed to enable nodes to base their decisions on the transmission schedules of other nodes. The limitations of deep RL approach includes lengthy adaptation time, the complexity and computational requirements of each node, and the lack of guarantees. In [45], only networks with at most two deep RL nodes have been demonstrated, and even so, the adaptation time is to the order of 10,000 time slots.

A less powerful, but faster learning, type of reinforcement is *expert-based* learning, in which nodes learn which of different "experts", or transmission strategies, to follow [17, 18, 6]. In ALOHA-Q, [10, 9], (framed slotted ALOHA with Q-learning), proposed by Chu et al., nodes used Q learning to choose in which slot of a fixed-length transmission frame to transmit. The protocol assumes a fixed frame length M. Each node has the set of policies $\{(i, M) \mid 0 \leq i < M\}$, where policy (i, M) prescribes transmitting in the i-th time-slot of the frame, that is, at all time slots t where $t \mod M = i$. Whenever a node transmits a packet in time slot t, it updates the weight of policy $(t \mod M, M)$, increasing it if the transmission succeeds, and decreasing it otherwise. At each frame, each node transmits according to the policy of highest weight, with a back-off procedure if a collision occurs. When the number of active nodes approaches N, the overall network throughput after adaptation can approach 1. Unfortunately, when the number of active nodes is below N, some bandwidth goes unused. Furthermore, when the number of active nodes exceeds N, collisions are bound to occur, and ALOHA-Q reverts to backoff strategies that ultimately degrade the performance to the $1/e$ limit as the number of active nodes grows.

Policy Tree based ALOHA, introduced in this thesis, is similar to ALOHA-Q in being rooted in expert-based RL. Unlike ALOHA-Q, our work eliminates the use of the fixed-length transmission frames that are assumed in all prior approaches aimed at improving slotted ALOHA. Rather, the "experts" of policy tree based consist in a tree of periodic schedules. The schedules have periods that are powers of two: the root schedule transmits at all times (period $2^0 = 1$), and the two children of a schedule each transmit with period of $2^1 = 2$, and half of the transmissions as the parent schedule. The policy trees based ALOHA directly recall the conflict resolution scheme of Capetanakis et al. [7]. However, while the scheme of Capetanakis et al. aimed at resolving each conflict as it arose, our tree schedule is used to let nodes learn transmission policies that avoid conflicts to begin with.

We point out that all of this prior work in RL based ALOHA relies on implicit, immediate acknowledgments of transmissions. The assumption is that a node immediately learns the outcome of its own transmission. This type of feedback happens, for instance, when there is a centralized repeater that rebroadcasts all received packets on a separate orthogonal channel. Two of the policy-tree based methods presented in this thesis (chapter 5 and chapter 6) allow for delayed acknowledgment, where after transmitting, a node does not know the transmission outcome until a few time slots later, when they hear back an acknowledgement from other nodes. This makes the protocols useful in ad-hoc settings where peer-to-peer transmission is enough to establish consensus without needing centralized control or centralized coordination.

Chapter 3

Adaptive Tree ALOHA

3.1 AT-ALOHA

3.1.1 Overview

Adaptive Tree ALOHA (AT-ALOHA) nodes learn to cooperatively share a time-slotted transmission channel [46]. The operation of AT-ALOHA is very much the same as that of slotted ALOHA, except for the transmission strategy used by a node and the mechanism with which it updates the transmission strategy.

In AT-ALOHA, at each time slot a node has the choice of either transmitting or waiting. We denote these two actions by T and W, respectively. If all nodes with packets to send wait, then the time slot is empty; if exactly one node transmits, then the slot has a successful transmission; if more than one node transmits, then a collision occurs. We denote these three outcomes by E, S, and C, respectively.

Policies, or ways in which AT-ALOHA nodes decide to transmit, consist of

the union of simple periodic schedules. Periodic schedules are chosen because a natural way to achieve coordination is to take turns, and details of policies and schedules are described in the subsequent section.

3.1.2 Schedules and Policies

We assume that every node has a clock t that counts the number of time slots. A (periodic) *schedule* (i, m) prescribes transmitting at all times t such that $t \mod 2^m = i$, in other ways, transmitting at time slot i every 2^m time slots; we let $T(i, m) = \{t \mid t \mod 2^m = i\}$ be the set of transmission time slots of (i, m). Let $\mathcal{S} = \{(i, m) \mid m > 0, 0 \leq i < 2^m\}$ be the set of all such periodic schedules. The schedules in $\mathcal{S}$ can be arranged in a tree, where the schedule (i, m) has $(i, m + 1)$ and $(i+2^m, m+1)$ as children: a child schedule transmits in half the time slots as its parent. See Figure 3.1 for an example of policy tree, in which the policy tree consists of two schedules (the two dark nodes).

Note that, AT-ALOHA does not constrain nodes to transmit according to a single schedule. Doing so would result in the bandwidth of individual nodes assuming only values corresponding to the fractional powers of 2: $1, 1/2, 1/4, 1/8, \ldots$. Instead, AT-ALOHA uses a *policy* $\pi \subseteq \mathcal{S}$ consisting of a set of schedules; the transmit times of π are the union of the transmit times of the individual schedules in π, or $T(\pi) = \cup_{s \in \pi} T(s)$.

AT-ALOHA policies are required to be in *normal form*, in order to keep the tree minimal, and to prevent it from growing unnecessarily large and fragmented:

- *No descendants:* if $s, s' \in \pi$, then s' is not a descendant of s in the policy tree.

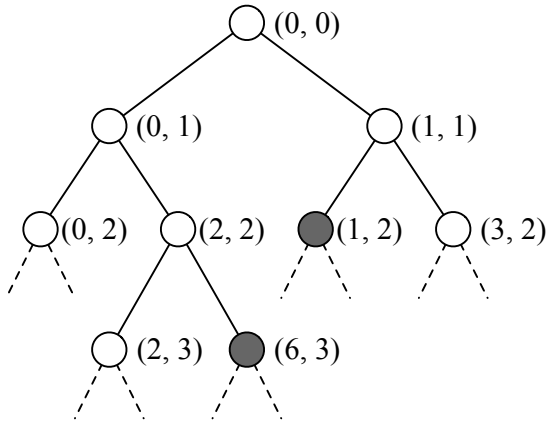


Figure 3.1: **An Example Policy Tree**. The dark nodes correspond to policy $\pi = \{(1,2),(6,3)\}$, which represents transmission at time slot 1 every $2^2 = 4$ slot and also at time slot 6 every $2^3 = 8$ slots.

- *No siblings:* for all $m > 0$, and $0 < i < 2^m$, it is never the case that both $(i, m+1) \in \pi$ and $(i + 2^m, m+1) \in \pi$.

The policy in Figure 3.1 is in normal form. Every policy can be put in normal form without affecting its transmission times by first eliminating descendant schedules, and then by merging all sibling schedules $(i, m+1)$ and $(i + 2^m, m+1)$ into (i, m) (repeating the merging until no siblings remain). The set of AT policies consists of all finite sets of schedules that are in normal form; we denote it by $\mathcal{P}$.

3.1.3 The AT Algorithm

Algorithm 1 specifies the AT algorithm.

AT updates the time-slot counter t, the policy $\pi \in \mathcal{P}$, and two probabilities p_b and p_k, known as the *barge-in* and *kindness* probabilities, whose role we describe in the following.

The policy, time-slot counter, and probabilities are local to each node; in particular, nodes do not need to agree on the numbering of time slots, and simply start counting time slots when joining the protocol. In fact, policy π associated with a time-slot counter t is equivalent to a policy shift$(\pi, \Delta) = \{((i + \Delta) \mod 2^m, m) \mid (i, m) \in \pi\}$ associated with counter $t + \Delta$.

The algorithm can be in two states, *active* and *inactive,* depending on whether the nodes has packets that need sending or not. A node makes a decision d to transmit (T) or to wait (W) according to this criteria: if it is active and $t \in T(\pi)$, then $d = T$, otherwise $d = W$. The node then receives the channel state $c \in \{E, C, S\}$ (denoting empty, collision or success), and uses it to update π, p_b, and p_q in following ways:

- If $c = C$ and $d = T$. Node has caused a collision. The schedule in π that caused the collision is either eliminated or replaced by one of its two children in the tree, chosen at random, reducing its bandwidth by at least half. This is implemented in the *demote* procedure.

- If $c = S$ and $d = T$, Node has successfully transmitted. With a small probability p_k (the kindness probability) the node demotes the policy responsible for a transmission, to allow for the fairness of rotation of slot use among nodes.

- If $c = E$ and $d = W$,there has been an empty slot. With probability p_b (the barge-in probability) the node adds a schedule to the policy π to make use of the free time slot. This is implemented in the *barge-in* procedure.

After this update, the policy π is pruned and brought back into normal form

via the *normalize* procedure, also described in detail later.

3.1.4 Policy Update Operations

AT-ALOHA policy update make uses of three operations: *demote, barge-in,* and *normalize.*

demote(π, t) (Figure 3.2). The procedure *demote*(π, t) removes all schedules (i, m) for which $t \in T(i, m)$ from its policy tree π. Further, if $\{(j, k) \in \pi \mid k \leq m\} = \emptyset$, then the procedure adds to π one of the two children $(i, m+1)$ or $(i+2^m, m+1)$ of the removed schedule, chosen uniformly at random.

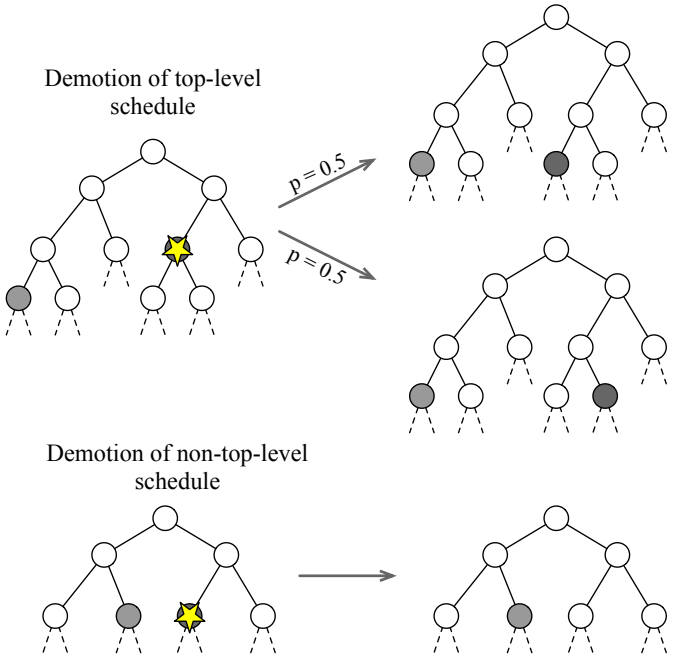


Figure 3.2: **Policy Update - Demote**. The starred schedule in the policy tree is demoted, in two scenarios according to whether the demoted schedule is at top level in the policy or not.

bargein(π, t, Δ_{new}). The procedure *bargein*(π, t, Δ_{new}) adds to policy π a new schedule

Constants:

$\alpha_k = 0.98$: kindness inertia;
$\alpha_b = 0.99$: barge-in inertia;
$q_k = 10^{-2}$: kindness probability lower bound;
$q_b = 10^{-3}$: barge-in probability lower bound;
$\kappa = 0.05$: target fraction of empty slots;
e: base of natural logarithm;
$M = 10$: maximum number of schedules in a policy;
$\Delta = 4$: maximum schedule level difference;
$\Delta_{new} = 2$: schedule insertion delta;

State Variables:

active: True if the node is active; false otherwise;
t: time slot counter;
π: AT policy;
p_b, p_k: burst-in and kindness probabilities;

Channel Variables:

T : transmit; W : wait;
$d \in \{T, W\}$: decision;
S : successful time slot;
E : empty time slot;
C : time slot with collisions;
$c \in \{S, E, C\}$: channel state;

Initialization:

$t := 0$; $p_b := 0.1$; $p_k := 0.05$;
$\pi := choice\{(0, 1), (1, 1)\}$;

At every time slot:

$t := t + 1$;
`// Decision, and outcome`
if $t \in T(\pi)$ *and active* **then** $d := T$ **else** $d := W$;
$h :=$ *channel outcome in* $\{E, C, S\}$;
`// Policy update`
if $d = T$ **then**
  if $h = S$ **then**
    with probability p_k**:**
      $\pi := demote(\pi, t)$
  if $h = C$ **then** $\pi := demote(\pi, t)$;
if $d = W$ *and* $h = E$ **then**
  with probability p_b**:**
    $\pi = bargein(\pi, t, \Delta_{new})$
$\pi := normalize(\pi, M, \Delta)$;
`// Probability update`
if $h = E$ **then** $p_k := p_k \cdot \alpha_k^{1/\kappa}$ **else** $p_k := p_k/\alpha_k$;
if $h = E$ **then** $p_b := p_b/\alpha_b$;
if $h = C$ **then** $p_b := p_b \cdot \alpha_b^{1/(e-2)}$;
$p_k := \min(0.5, \max(q_k, p_k))$; $p_b := \min(0.5, \max(q_b, p_b))$

Algorithm 1: AT-ALOHA Algorithm.

(i, m) such that $t \in T(i, m)$. Let $b = \sum_{(i,k)\in\pi} 2^{-k}$ be the bandwidth of the node's policy. We let

$$m = \left\lceil \log_2(1/p_b) + [\log_2(b/p_b)]_{-1}^{1} + \Delta_{new} \right\rceil \ , \tag{3.1}$$

(where $[x]_{-1}^{1}$ means to clip the value of x at maximum of 1 and minimum of -1) and $i = t \mod m$; the choice of m is discussed in subsection 3.1.5.

Figure 3.3 shows an example of barge-in.

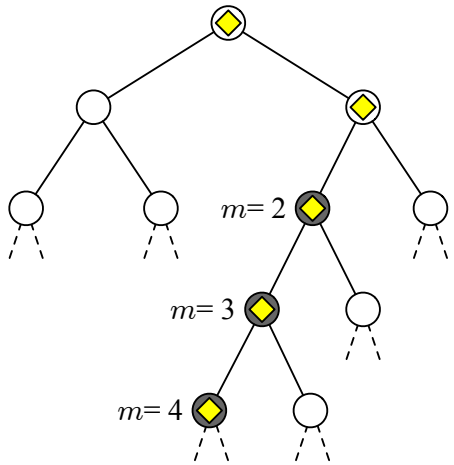


Figure 3.3: **Policy Update - Barge-in**. The schedules indicated with a diamond are candidates for schedule insertion into the policy tree, as they would have transmitted in the empty slot. One of the schedules at level $m \in \{2, 3, 4\}$, is inserted.

normalize(π, M, Δ)***:*** To normalize a policy π, the following steps are repeated until they can no longer be taken:

- *Descendant elimination:* If there are: $(i, m), (j, k) \in \pi$ with $k > m$ and $j \mod 2^m = i$, remove (j, k) from π.
- *Siblings merging:* If there are $(i, m), (j, m) \in \pi$ with $j = i + 2^{m-1}$, then replace both (i, m) and (j, m) in π with $(i, m - 1)$.

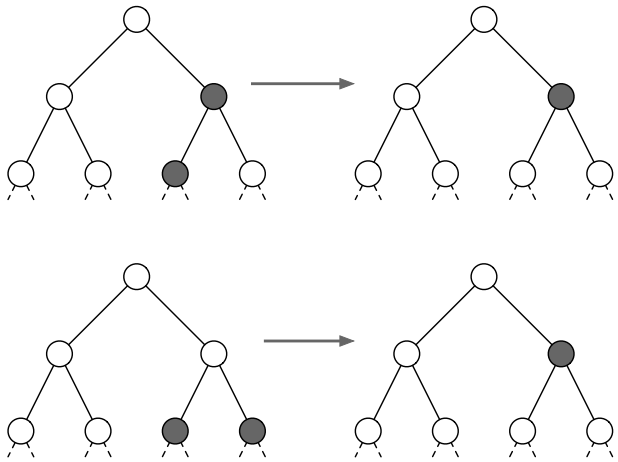


Figure 3.4: **Policy Normalization**. Two policy normalization operations: descendant elimination (top) and sibling merging (bottom).

Figure 3.4 illustrates these two normalization operations. Once π is in normal form, we prune it in two steps, first limiting the tree depth, then the number of selected schedule nodes in it. These pruning operations, illustrated in Figure 3.5, are performed as follows:

- Let $k = \min\{m \mid (i,m) \in \pi\}$ be the minimum level of a schedule in π. AT prunes all schedules of level below $k + \Delta$, letting $\pi := \{(i,m) \mid (i,m) \in \pi \wedge m \leq k + \Delta\}$.

- AT then prunes π to ensure it contains at most M schedules. If $|\pi| \leq M$, AT leaves π unchanged. Otherwise, let $n_k = |\{(i,m) \in \pi \mid m \leq k\}|$, and let k be the largest integer such that $n_k \leq m$. Then, AT removes from π all schedules (i,m) with $m > k + 1$, and we randomly select $M - n_k$ of the schedules at level $k + 1$, that is, of the form $(j, k + 1)$ for some j.

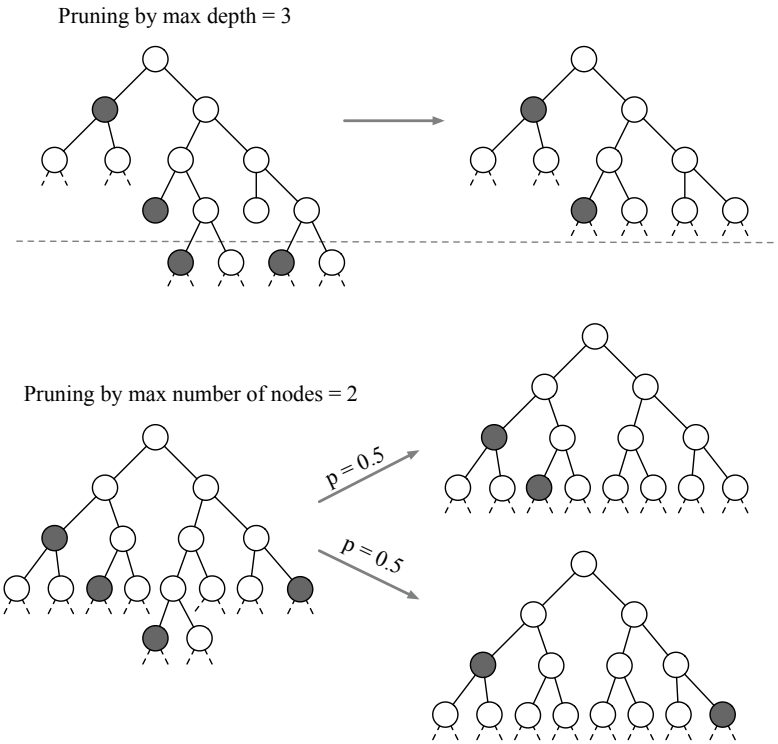


Figure 3.5: **Policy Pruning**. Pruning a policy by limiting the depth (top), and limiting the number of nodes (bottom).

3.1.5 Fairness, Kindness, and Barge-in

The kindness and barge-in probabilities p_k and p_b, together, ensure that every active node receives a fair share of the total bandwidth. The kindness probability ensures that a node has a non-zero probability of relinquishing any transmission slots it holds. Nodes that transmit in more slots relinquish proportionately more bandwidth than nodes using fewer slots, and every freed slot has the same probability of being captured by any node. Together, this ensures that the bandwidth tends to be uniformly distributed among the nodes participating in the protocol. The probabilities p_k and p_b are tuned dynamically as follows.

Tuning of Kindness Probability The kindness probability p_k is tuned so that for each n successful slots, we have κn of free slots; we choose $\kappa = 0.05$ or 5%. Initially we arbitrarily set $p_k = 0.05$. Thereupon, nodes update p_k according to the channel outcomes E, S, C:

$$E: \ p_k := p_k \cdot \alpha_k^{1/\kappa} \qquad S, C: \ p_k := p_k/\alpha_k$$

where $\alpha_k = 0.98$ is a coefficient determining the adaptation speed. Thus, p_k increases at each success or collision slot, and decreases at each empty slot. Equilibrium is reached when these updates balance, that is, when there are κ empty slots for every successful or collision slot, so that $(\alpha_k^{1/\kappa})^\kappa = \alpha_k$.

Tuning of Barge-in Probability The barge-in probability is tuned to optimize the probability that when a slot is empty, one node, and only one node, will add a schedule to use the empty slot in the future. The analysis follows the lines of the slotted ALOHA analysis in [39]. If there are n active nodes and each of them barges-in with probability q, then a time slot remains empty with probability $(1-q)^n$, it is used successfully with probability $nq(1-q)^{n-1}$, and collision happens with probability $1-(1-q)^n - nq(1-q)^{n-1}$. The probability of successful transmission is maximized for $q = 1/n$ when n nodes are active. Under this optimal choice of q, as $n \to \infty$, the probability of the slot remaining free tends to $1/e$, and the collision probability tends to $2/e$, where e is the basis of the natural logarithm. The optimal ratio of free to collision slots is then $(1/e)/(1-2/e) = 1/(e-2)$. Thus, the nodes tune p_b so that the ratio of

free to collision slots is $1/(e-2)$, updating p_b according to channel outcomes:

$$E:\ p_b := p_b/\alpha_b \qquad C:\ p_b := p_b \cdot \alpha_b^{1/(e-2)}\ .$$

For the adaptation coefficient, we take $\alpha_b = 0.99$.

We have seen that the barge-in probability p_b provides an estimate $\tilde{n} = 1/p_b$ of the number of active nodes in the protocol. After experimentation, we choose to insert new schedules at around level $m = \log_2(\tilde{n}) + \Delta_{new}$ with $\Delta_{new} = 2$.

3.2 Performance

We compare AT-ALOHA with framed slotted ALOHA and ALOHA-Q via simulations of a fully-connected single-channel time-slotted wireless network. The three protocols are compared in terms of their bandwidth utilization and fairness.

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 takes the decisions of all nodes for every time slot, computes the outcome (empty, successful transmission, or collision), and relays the outcome to each node. The node modules implement each protocol algorithm at each node. For AT-ALOHA, for instance, the node module implements the time-slot counter, the policy tree, and Algorithm 1. Protocol modules for EB-ALOHA and ALOHA-Q can be similarly implemented.

Network utilization A time slot is either empty or contains a successful transmission or a collision. To show how the network utilization evolves over time, we aggregate time slots in *blocks* of 100, and for each block we can compute the utilization as a fraction of individual slots that contains a successful transmission. Similarly, we can measure the fraction of empty and collision time slots in each block. Using blocks of length 100 offers a compromise between having a fine time resolution, and computing meaningful statistics on each block.

Fairness We measure the fairness of the protocols via *Jain's index* [20, 19]. For a block in which n nodes are active, let b_i be the number of successful transmission by node i, for $i \leq i \leq n$. Jain's index is computed as

$$J = \frac{(\sum_{i=1}^{n} b_i)^2}{n \sum_{i=1}^{n} b_i^2} \tag{3.2}$$

We have $1/n \leq J \leq 1$; $J = 1$ for a perfectly fair distribution of the channel, and $J = 1/n$ if only one node gets to use the channel. To compute Jain's index, we consider groups of 10 blocks (i.e., 1000 time slots), to minimize variations due to statistical fluctuations in the number of transmissions per block.

3.2.1 Comparison Protocols

We compare the performance of AT-ALOHA with that of two versions of exponential-backoff ALOHA, and with the performance of the ALOHA-Q protocol proposed by Chu et al. [10, 9].

EB-ALOHA and EB-ALL-ALOHA EB-ALOHA is the standard slotted ALOHA with exponential-backoff. In EB-ALOHA, every node, when becoming active, has an initial transmission probability $p = 1/2$. Whenever the node transmits, it updates the transmission probability, setting $p := \alpha p$ in case of collision, and $p := \min(1, p/\alpha)$ in case of success, where α is a constant that determines adaptation speed; in our simulations we use $\alpha = 0.9$. The EB-ALL-ALOHA protocol is similar to EB-ALOHA, except that nodes update their transmission probabilities following *all* successful transmissions or collisions, rather than only those in which they took part.

ALOHA-Q ALOHA-Q is the Q-learning version of ALOHA proposed in [10, 9]. The ALOHA-Q is based on a periodic frame of fixed length n. Each node stores *q-values* $q_1, q_2, \ldots, q_n$, where q_i represents the quality of the decision of transmitting in the i-th slot of the frame. At every frame, the protocol transmits in a slot i with maximal q_i; if the transmission is successful, it increases q_i; if a collision occurs, it decreases q_i and it follows a randomized backoff before retrying. The bandwidth utilization of ALOHA-Q increases with the number m of active nodes, approaching m/n, as long as $m \leq n$; when $m \gg n$, the protocol behaves in a similar fashion to EB-ALOHA. In our simulations, we consider frames of $n = 64$ time slots; as our maximum value for m is 50, this ensures that the protocol works close to optimality.

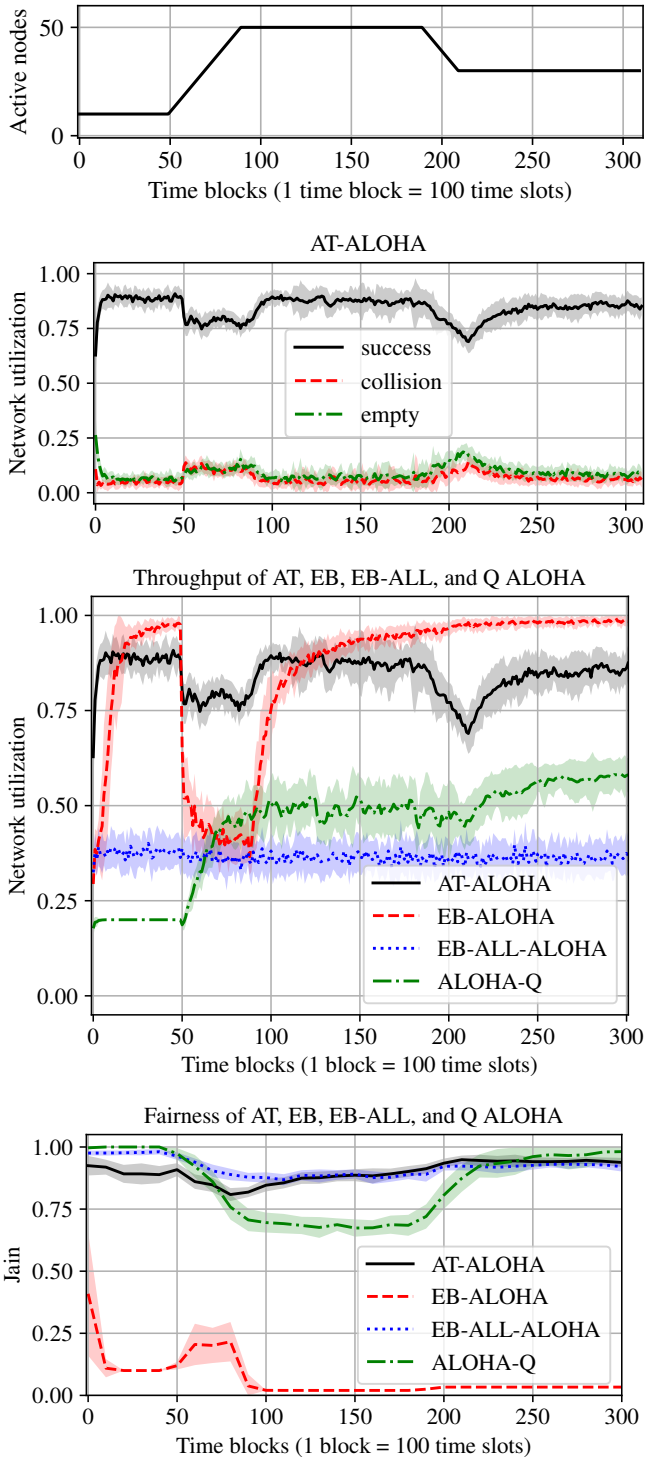


Figure 3.6: **Performance of AT-ALOHA**. AT-ALOHA Protocol compared with ALOHA-EB and ALOHA-Q, when the number of active nodes is fist 10, then ramps up to 50, and finally ramps down to 30.

3.2.2 Results

Figure 3.6 compares the performance of AT-ALOHA, EB-ALOHA, EB-ALL-ALOHA, and ALOHA-Q when the number of active nodes is initially 10, then ramps up to 50, and finally ramps down to 30. The solid line shows the average of 20 runs and the shaded region represent plus/minus one standard deviation of the 20 runs.

For AT-ALOHA, the network utilization remains in the 85% to 90% range in the steady-state periods when nodes neither join nor leave; during the transients, the utilization is still above 75% when ramping up, and above 60% when ramping down. The Jain fairness index of AT-ALOHA is also close to 1.

The only protocol that is competitive with AT-ALOHA in terms of utilization is EB-ALOHA. The problem is that EB-ALOHA achieves its high network utilization via an extremely unfair allocation of bandwidth, leading to a Jain index close to 0. In EB-ALOHA, nodes that are successful in transmitting will increase their transmission probability, while nodes whose transmissions are unsuccessful due to collisions will reduce their transmission probability. This amplifies any initial random difference in transmission success, leading to a winner-takes-all situation in which one node uses most of the bandwidth, transmitting with very high probability, while other nodes are mostly silent.

The EB-ALL-ALOHA protocol manages to achieve the optimal network utilization of $1/e \approx 0.37$ that is the maximum attainable under symmetrical transmission probability (and thus fairness) for ALOHA. Its fairness is uniformly very high, since all

nodes transmit with the same probability.

Finally, the network utilization of ALOHA-Q is dependent on the number of active nodes, increasing as the number of active nodes approaches the frame length of 64. Even when the number of active nodes is 50, as around time block 150 of Figure 3.6, the utilization is below 0.6. This is well below the theoretical maximum of $50/64 \approx 0.78$, likely because the active nodes have not had time to adapt to the network conditions. ALOHA-Q also allocates bandwidth fairly, as each node can transmit at most once per frame.

3.3 Conclusion

We introduced a new collaborative learning algorithm, the Adaptive Tree (AT) algorithm, to enable nodes sharing a common channel to quickly approach collision-free transmissions while maintaining fairness. In contrast to prior approaches that use machine learning to improve the performance of slotted ALOHA, the resulting protocol, AT-ALOHA, only requires nodes to agree on the beginning of time slots, and does not require the definition of transmission frames with a fixed number of time slots per frame or the numbering of time slots. Simulation experiments were used to illustrate that AT-ALOHA attains better throughput and fairness than slotted ALOHA with exponential backoffs and ALOHA-Q, which is framed slotted ALOHA with Q learning.

Chapter 4

Quantitative Tree ALOHA

4.1 ALOHA-QT and ALOHA-QTF

4.1.1 Overview

ALOHA-QT and ALOHA-QTF represent a different approach than AT-ALOHA to solve the same problem. In AT-ALOHA, each node remembers a subset of active schedules, which consist in the node's schedule. In ALOHA-QT and ALOHA-QTF, each node remembers the *weights* of every possible schedule, and dynamically selects for use the schedules with highest weight.

In ALOHA-QT and ALOHA-QTF [12], the network setup is the same as AT-ALOHA. We consider a fully-connected network in which the channel is time slotted. At each time slot a node can either transmit (T) or wait (W), and the channel can be in one of three states: empty (E), if no node transmits; success (S), if exactly one node transmits; and collision (C), if two or more nodes transmit. We assume that

the nodes can immediately detect the state of the time slot, and thus, the outcome of their transmissions. This assumption can be brought to practice in several ways. A central node (such as a satellite transponder or a base station) can use a downlink to repeat the transmissions sent to it in each slot over an uplink, as in the original slotted ALOHA protocol. Alternatively, each time slot can be divided into a portion for packet transmission and a portion for an acknowledgement.

It is well known that the throughput of slotted ALOHA in a fully-connected network tends to $1/e \approx 0.37$ as the number of nodes grows. Achieving higher throughput while giving each node a fair share of the bandwidth requires coordinating the node's transmission schedules. ALOHA-QT and ALOHA-QTF attempt to do this by allowing node to *learn to coordinate* via reinforcement learning, without need for any centralized coordination, pre-agreement, or out-of-band communication.

4.1.2 From ALOHA-Q to ALOHA-QT and ALOHA-QTF

The starting point in our design is ALOHA-Q [10, 9], a protocol based on a fixed-length frame, in which each node learns in which slot of the frame the transmissions can be most successful. ALOHA-Q is based on a transmission frame of fixed length M. An ALOHA-Q node keeps a time-slot counter t, and relies on M schedules $(0, M)$, $(1, M)$, $\ldots$, $(M-1, M)$; a schedule (i, M) prescribes sending at all times t such that $t \mod M = i$. The schedules play the role of the experts in reinforcement learning: each node tracks the success of each expert,, and so eventually settles into transmitting always in the same time-slot of each frame.

ALOHA-Q has two limitations. One is its reliance on a fixed-length frame. When the number of nodes N is smaller than the frame length M, the network utilization can approach N/M in the long term. However, when $N \ll M$ the utilization can be very low, and when $N > M$, the utilization is quickly degraded, as there are not enough slots for each node in the fixed-length period. The second limitation of ALOHA-Q is the adoption of a non-standard version of weight update for the schedules, which slows down adaptation.

Our approach improves on ALOHA-Q in four main ways. The first, and most important, is to abandon the use of a fixed-length frame, and adopt instead a *policy tree*, where transmission schedules of different periods are arranged in a tree. The tree guides conflict resolution and helps the nodes settle on non-conflicting schedules. The availability of schedules of different periods enables the nodes to vary their transmission rate and adapt to the number of active nodes: there is no longer a fixed frame that can remain mostly empty, or be of insufficient length to accommodate all nodes. The second, related, improvement is to allow nodes to follow the "advice" of more than one expert simultaneously; this lets nodes mix schedules of different transmission bandwidth, enabling the nodes to fine-tune their overall transmissions. The third improvement consists in a new method for updating the policy tree weights following network outcomes, which enables a faster convergence to an efficient schedule. The last improvement consists in tuning the weight updates and schedule selection to ensure a fair distribution of the network bandwidth to the participating nodes.

4.1.3 The Quantitative Policy Tree

Each network node keeps a time-slot counter t. Like AT-ALOHA in Chapter 3, the transmission schedules are organized in a policy tree. Each (periodic) *schedule* consists in a pair (i, m) with $0 \leq i < m$; the schedule (i, m) prescribes transmitting at all times t such that $t \mod m = i$. For instance, the schedule $(3, 8)$ prescribes transmitting at times $t = 3, 11, 19, 27, \ldots$. Importantly, our design does not require the nodes to synchronize their time-slot counters: a schedule (i, m) for a node with counter t is equivalent to a schedule $((i + k) \mod m, m)$ for a node with counter $t' = t + k$.

The reinforcement learning at each node is based on the set of schedules $\mathcal{P} = \{(i, 2^m) \mid 0 \leq i < 2^m, 0 \leq m \leq n\}$. The maximum periodicity 2^n (which represents schedules at the bottom of the tree) is chosen so that it is larger than the maximum number of nodes that can be present on the network. Since nodes can transmit with periods that are smaller than 2^n (for example, nodes can transmit every fourth time-slot using a schedule of period 2^2), there is no performance penalty in ALOHA-QT in choosing a value of n that is larger than necessary.

In principle, reinforcement learning could be applied to arbitrary sets of schedules, such as $(1, 5)$, or $(2, 3)$, representing transmission once very 5th time slot or once every 3rd time slot. However, constraining the periods to be powers of 2 facilitates the coordination between the nodes. To illustrate this, the transmission schedules of ALOHA-QT are arranged into a binary *policy tree*, where the root of the policy tree is the schedule that transmits at all times, and where the two children of a schedule

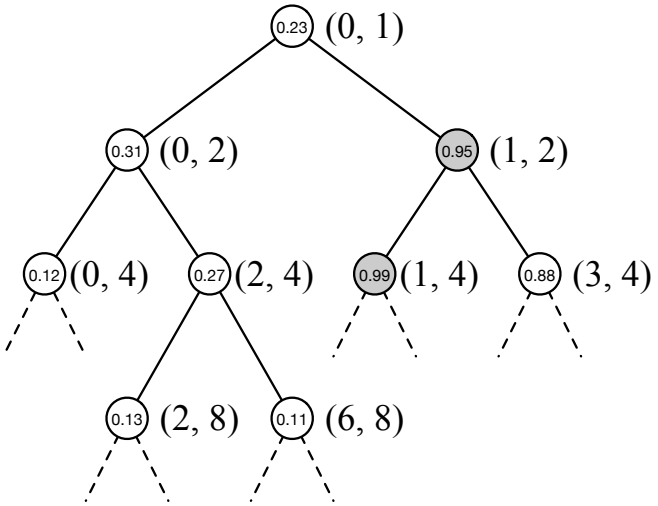


Figure 4.1: **Quantitative Policy Tree in ALOHA-QT and ALOHA-QTF**. Numbers next to each tree node (i, m) represent its transmitting schedule (i every m time slot). Each tree node/schedule also stores a weight value (shown inside the circle). The schedules with the highest weights (grey) determine the overall transmission schedule of the protocol.

node transmit each in half of the time slots of the parent, as depicted in Figure 4.1. The schedule $(0, 2)$ in the tree prescribes transmitting at even time slots, and has as children the two schedules $(0, 4)$ and $(2, 4)$ which both prescribe transmission once every four time slots; the union of the transmission schedule of these two children schedules is exactly the set of transmission schedule of the parent $(0, 2)$. In this tree, two schedules prescribe conflicting transmissions only if one schedule is an ancestor of the another. Thus, as long as nodes settle on schedules that are not the ancestor of the other, the nodes can transmit on the network avoiding conflicts. If we allowed schedules with arbitrary periods, rather than schedules in the tree with periods that are powers of 2, conflicts would be common: for instance, if two schedules (i, m), (i', m') have mutually prime periods m, m', then there would be a collision every mm' time-slots.

4.2 The ALOHA-QT Algorithm

Algorithm 2 describes how each node selects time slots for transmission in ALOHA-QT. Each node keeps a time-slot counter t, and it stores the weight $w_\sigma \in [0, 1]$ of each schedule σ in the policy tree. At each time slot, each node performs the following actions:

- *Schedule selection.* The node selects a subset $\mathcal{A}_t$ of *active* schedules to follow in the time slot t, based on the weights of the schedules.
- *Decision.* If $t \mod m = i$ for some active schedule $(i, m) \in \mathcal{A}_t$, and the node is active (it has some packet to send), the node transmits; otherwise, it waits.
- *Weight update.* Based on the resulting channel state (Successful, Empty, Collision) of the time slot, the node updates the weights of all schedules.
- *Weight normalization.* Once the weights of the schedules have been individually updated, the values for all schedules are normalized, redistributing some of the "lost weight" randomly across all schedules.

4.2.1 Weight Initialization

Let n be the depth of the policy tree. For $k = 1, 2, \ldots, n$ and $0 \leq i < 2^k$, we initialize the weight of schedule $(i, 2^k) \in \mathcal{P}$ by:

$$w_{(i,2^k)} = 0.2 \cdot \frac{0.9 + 0.1 \cdot Z_{(i,2^k)}}{1.2^k} ,$$

Constants:

$n = 8$: depth of policy tree;
$w_{init} = 0.25$: weight initialization factor;
$\alpha^+ = 0.2$: multiplicative increment factor;
$\alpha^- = -0.5$: multiplicative decrement factor;
$\gamma_0 = 0.1$: weight initialization noise;
$\gamma_1 = 1.2$: initial bias for high-bandwidth schedules;
$\eta = 0.95$: weight selection threshold;
$\epsilon_r = 0.02$: probability of relinquishing a time-slot;

State Variables:

$\mathcal{P} = \{(i, m) \mid 0 \leq i < m, m = 2^k, 0 \leq k \leq n\}$: schedules;
$\{w_\sigma\}_{\sigma \in \mathcal{P}}$: schedule weights;
active: True if the node is active; false otherwise;
$t \in \mathbb{N}$: time slot counter;

Channel Variables:

$d \in \{T, W\}$: decision (T : transmit; W : wait);
$c \in \{S, E, C\}$: channel state (S : successful time slot, E : empty time slot, C : time slot with collisions);
X: random sample from the uniform distribution over $[0, 1]$;

Initialization:

$t := 0$;
for $0 \leq k \leq n$, $0 \leq i < 2^k$ **do**
    $w_{(i,2^k)} = w_{init} \cdot \gamma_1^{-k} \cdot (1 - \gamma_0 + \gamma_0 \cdot X_{ik})$;

At every time slot:

`// schedule selection and decision`
$\mathcal{E}_t = \{(i, m) \in \mathcal{P} \mid i \mod m = t\}$;
$\mathcal{A}_t = \text{argmax}_{\sigma \in \mathcal{P}} w_\sigma \cup \{\sigma \in \mathcal{P} \mid w_\sigma > \eta\}$;
if $\mathcal{E}_t \cap \mathcal{A}_t \neq \emptyset$ *and active* **then** $d := T$ **else** $d := W$;
`// weight update`
$h :=$ *channel outcome in* $\{E, C, S\}$;
if $(d, h) \in \{(W, E), (T, S)\}$ **then** $\alpha := \alpha^+$ **else** $\alpha := \alpha^-$;
for $\sigma \in \mathcal{P}$ **do**
    if $\sigma \in \mathcal{E}_t$ **then** $w'_\sigma := w_\sigma \cdot e^{\alpha X_\sigma}$ **else** $w'_\sigma = w_\sigma$;
`// Voluntary slot relinquishment`
if $X < \epsilon_r$ **then**
    for $\sigma \in \mathcal{E}_t$ **do** $q'_\sigma = 0$;
`// Weight normalization`
$W := \sum_{\sigma \in \mathcal{P}} w_\sigma$, $W' := \sum_{\sigma \in \mathcal{P}} w'_\sigma$, $\Delta = W - W'$;
if $\Delta > 0$ *and* $W' < W_{init} \cdot |\mathcal{P}|$ **then**
    for $\sigma \in \mathcal{P}$ **do** sample X_σ;
    for $\sigma \in \mathcal{P}$ **do** $w_\sigma := w'_\sigma + \Delta * (X_\sigma / \sum_\sigma X_\sigma)$;
else
    for $\sigma \in \mathcal{P}$ **do** $w_\sigma := w'_\sigma$;
for $\sigma \in \mathcal{P}$ **do** $w_\sigma := \min(1, w_\sigma)$;
`// Increment time`
$t := t + 1$;

Algorithm 2: ALOHA-QT Algorithm.

where $\{Z_\sigma\}_{\sigma \in \mathcal{P}}$ is a set of random variables independently sampled from the uniform distribution over $[0, 1]$. Thus, schedules have an initial weight of approximately 0.2, with a small amount of noise added to break ties between schedules and to ensure that the initial behavior of nodes is not synchronized. The denominator 1.2^k causes schedules with shorter periods to have higher probability of being initially active. In this fashion, nodes are initially likely to adopt schedules that transmit frequently, falling back on schedules that transmit more rarely only as needed to avoid collisions.

4.2.2 Schedule Selection

We denote by $\mathcal{A}_t$ the set of schedules that have been selected as active at time t. A schedule $\sigma \in \mathcal{P}$ is selected as active in a round if:

1. either σ is the schedule with the maximal weight w_σ among all schedules in a tree;
2. or $w_\sigma \geq w_h$, where w_h is a pre-determined weight threshold; we use $w_h = 0.95$ in our implementation.

The first criterion ensures that a node always follows its best schedule: this guarantees that every node will transmit at least once every 2^n time-slots. The second criterion allows a node to follow any additional schedule that has been successful in predicting available slots. The ability of nodes to follow more than one schedule is instrumental in enabling nodes to utilize a flexible amount of network bandwidth.

We experimented with alternative selection schemes to the one mentioned here; for instance, we experimented with selecting the highest-weight schedule σ, along with

any other schedule σ' such that $w_{\sigma'} > \alpha w_\sigma$, for a weight fraction $\alpha < 1$. These schemes did not work as well as the one we presented above.

4.2.3 Decision

We say that a schedule (i, m) is *enabled* in a time slot t if $t \mod m = i$: thus, the schedules enabled at a time slots are those that prescribe transmitting in time slot. We let $\mathcal{E}_t = \{(i, m) \in \mathcal{P} \mid t \mod m = i\}$ be the set of enabled schedules at time t. A node transmits (takes decision T) if $\mathcal{A}_t \cap \mathcal{E}_t \neq \emptyset$, that is, if one of the active schedules at t is enabled; otherwise, it waits (decision W). At the end of the time slot, the node receives the network state for the slot, which can be E (empty slot), S (successful transmission), or C (collision). The pair (d, h), consisting of the decision $d \in \{T, W\}$ and the network state $h \in \{E, S, C\}$, is called the *outcome* of the slot for the node.

4.2.4 Weight Update

At the end of a time slot, we apply a multiplicative update to the schedules that are enabled in the time slot, increasing their weight if transmitting does not lead to collisions, and decreasing if it does. The multiplicative update to the weights w_σ of all enabled schedules $\sigma \in \mathcal{E}_t$ takes the following form:

$$w'_\sigma = w_\sigma \cdot e^{X_\sigma \alpha} , \tag{4.1}$$

where:

- $\{X_\sigma\}$ is a set of random variables independently sampled from the uniform distribution $[0, 1]$, a new sample is drawn whenever X_σ is called;
- α is an update constant that depends on the slot outcome; we use:
  - $\alpha = 0.2$ for $(d, h) \in \{(W, E), (T, S)\}$;
  - $\alpha = -0.5$ for $(d, h) \in \{(W, S), (W, C), (T, C)\}$;
- w'_σ is the schedule weight after the update.

Due to X_σ, the multiplicative update factors are randomized. This is different from the common case for expert-based reinforcement learning [43, 18]. Randomization helps break the ties between nodes that lay claims on the same transmission slots. To understand this, consider the case of nodes transmitting in the same time slot, leading to a collision. If a deterministic update was used, the nodes would update the weights of the schedules responsible for the conflict in a synchronized manner, multiplying them by the same factor smaller than one. Eventually, the nodes involved in the collision would stop using the schedules and cease transmitting in the slot. As the slot became empty, the nodes would reverse course, and increase the weights of the conflicting schedules, likely reintroducing the conflict. This oscillatory behavior, from collisions to empty slots and back again, would slow down convergence to collision-free transmissions. Randomized updates break the symmetry and facilitate the emergence of a "winning" node that claims a slot; once a slot is claimed, the weight update mechanism reinforces the exclusive use of the slot by the winning node.

4.2.5 Weight Normalization

After the multiplicative update of the schedule weights, we perform a three-step normalization of the weights.

Relinquishing the slot First, with a small, constant probability ϵ_r, the weight of every schedule in $\mathcal{E}_t$ is set to 0, forcing the node to relinquish transmission at a time slot, if it were holding it. This ensures that no node can hold a slot forever, ensuring some amount of fairness in the bandwidth allocation to the nodes.

Redistributing lost weights In expert-based reinforcement learning, some of the weights lost by the experts that are downgraded is redistributed across all experts. In this way, if experts once successful become unsuccessful, the nodes will explore alternative experts [17, 18]. To this end, let w_σ, w'_σ be the weights of schedule σ before and after the multiplicative update step, let $W = \sum_{\sigma\in\mathcal{P}} w_\sigma$ and $W' = \sum_{\sigma\in\mathcal{P}} w'_\sigma$, and let $\Delta = W - W'$ 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 schedule, 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.

Bound enforcement Finally, the weights of all schedules is bound to the $[0, 1]$ interval, setting $w_\sigma = \min(1, w_\sigma)$. Bounding the weights of schedules that have been successful for a long time ensures that the weights can be reduced quickly, and the schedules abandoned, should the schedules become unsuccessful (that is, prescribe transmissions that cause collisions).

4.3 The ALOHA-QTF Algorithm

The bandwidth allocation of active nodes using ALOHA-QT is fair in the long term due to two reasons. First, every node has at least $1/2^n$ bandwidth, because a node always selects at least one schedule. Second, the more frequently a node transmits, the more frequently it will relinquish a time-slot, and once a time slot is relinquished, all nodes can lay a claim to it. Thus, in the long term the time slots will rotate the node to which they are allocated, and the overall allocation of bandwidth to the nodes will be fair.

This long-term fairness guarantee; however, it is not useful for nodes that are only active during short intervals of time in which they have data to send. We describe here a variation of the ALOHA-QT protocol, which we call ALOHA-QTF and achieves fairness in short time intervals. ALOHA-QTF differs from ALOHA-QT in two respects:

- nodes randomly relinquish time slots only if they use more than their fair share of bandwidth;
- the schedule weight update is made sensitive to the fraction of bandwidth used by

each node.

To implement these refinements, ALOHA-QTF keeps track of the number of recently active nodes in the network via a *participant counter*.

4.3.1 Counting Active Nodes and Estimating Fair Bandwidth

The ALOHA-QTF protocol achieves fairness by estimating the number of active network nodes, and by using the estimate to tune protocol parameters, among which the schedule weight updates. We assume that each node transmits its node ID (such as its MAC address) as part of each packet.

Counting active nodes To estimate the number of active network nodes, each node keeps a sliding window consisting of 2^n slots, where n is the depth of the policy tree. Each slot in the sliding window can contain either a node ID, or a placeholder $\perp$, which indicates no ID. At each network time-slot, the node deletes the left-most slot in the sliding window, corresponding to the oldest information, and adds to the right of the sliding window a new slot containing new information, according to the channel state $c \in \{S, E, C\}$:

- S: the ID of the node that transmitted successfully is added;
- E: $\perp$ is added;
- C: a random ID, taken uniformly at random from the space of node IDs, is added.

For example, if the sliding window contains $[\beta_1, \beta_2, \ldots, \beta_{2^n}]$, and a successful transmission by ID γ is received, the content of the sliding window is updated to $[\beta_2, \ldots, \beta_{2^n}, \gamma]$. Once this is done, the node produces the estimated $\hat{N}$ of the number of active network nodes by counting the number of distinct IDs (excluding $\perp$) in the sliding window. The rule for collisions ensures that, if there are network collisions, $\hat{N}$ tends to over-estimate the number of active nodes, as each collision is counted as a new participant (assuming the space of node IDs is much larger than the actual number of participating nodes, as is usually the case). This over-estimation works in the right direction, as it causes each node to trim down its transmissions, thus reducing the frequency of collisions.

Fair and requested bandwidth From the estimate $\hat{N}$ for the number of active nodes, the node computes the *fair* bandwidth $b_f = 1/\max(1, \hat{N})$: this is the bandwidth that each network node would receive under perfectly fair allocation. The node also computes *requested* bandwidth b_r, which is the fraction of network slots during which the node will transmit. To compute b_r, let $\delta(t, t') = 1$ if $\mathcal{A}_t \cap \mathcal{E}_{t'} \neq \emptyset$ and $\delta(t, t') = 0$ otherwise. In other words, $\delta(t, t') = 1$ if there is a schedule active at t which is scheduled to transmit at time t'. At the current time t, the requested bandwidth b_r is defined as

$$b_r = \frac{1}{2^n} \sum_{t'=0}^{2^n-1} \delta(t, t') \; .$$

The requested bandwidth can be computed efficiently by letting $\mathcal{B} := \mathcal{A}_t$, and by then removing from $\mathcal{B}$ all schedules that are the descendants, in the policy tree, of schedules already in $\mathcal{B}$. Precisely, we remove from $\mathcal{B}$ every schedule (i, m) such that there is $(i', m') \in$

$\mathcal{B}$ with $m' < m$ and $i \mod m' = i'$. For example, if $\mathcal{B} = \{(0,2),(0,4),(2,8),(3,4)\}$, the schedules $(0,4)$ and $(2,8)$ would be removed from $\mathcal{B}$, as they are descendants of $(0,2) \in \mathcal{B}$, leading to $\mathcal{B} = \{(0,2),(3,4)\}$. Once the set $\mathcal{B}$ is thus minimized, we have $b_r = \sum_{(i,m)\in\mathcal{B}} 1/m$.

4.3.2 Fair Weight Update

Once the fair and requested bandwidths b_f, b_r are computed, we modify the weight update in two ways.

- First, we apply the *Relinquish* step of Algorithm 2 only if $b_r > b_f$, that is, only if a node uses more than its fair share of bandwidth would it give up its claim on slots.

- Second, we modify the multiplicative update (4.1) by distinguishing the two cases of policy demotion ($\alpha < 0$) and policy promotion ($\alpha > 0$). For every $\sigma \in \mathcal{E}_t$, the update becomes, for $\alpha < 0$:

$$w'_\sigma = w_\sigma \cdot \exp\big(X_\sigma \cdot \alpha \cdot \min(1, (b_r/b_f)^{1/2})\big)\ , \tag{4.2}$$

  and for $\alpha > 0$:

$$w'_\sigma = w_\sigma \cdot \exp\big(X_\sigma \cdot \alpha \cdot \max(0, 1 - (b_r/b_f)^2)\big)\ . \tag{4.3}$$

The modified weight updates (4.2) and (4.3) can be interpreted as follows. In the case

of a weight reduction, or $\alpha < 0$, if $b_r < b_f$, that is, if the node is using less than its fair share of bandwidth, then the reduction is scaled down from 1 to $(b_r/b_f)^{1/2} < 1$. In this way, nodes that use less than their fair share see their schedule weights reduced less forcefully. Thus, when a node A that uses less than the fair share of bandwidth vies for the use of a time-slot with a node B that uses more than the fair share, the schedule weights of B will be reduced more, and A will tend to prevail in the use of the slot. Conversely, in the case of weight increase, or $\alpha > 0$, only nodes that request less than their fair share (that is, with $b_r < b_f$) will see their schedule weights increased. Thus, empty network slots will be more likely to be allocated to nodes whose active schedules request less than the fair share.

4.4 Performance Evaluation

We compare the performance of ALOHA-QT and ALOHA-QTF with the performance of ALOHA-Q [10, 9], and slotted ALOHA with exponential backoff (ALOHA-EB), by means of simulations. We consider a fully-connected single-channel wireless network in our comparisons. The channel is time slotted, and time slots are organized into transmission frames of 64 time slots each for the case of framed slotted ALOHA-Q. The length of a time slot equals a packet length, which is assumed to be a constant. The number of active nodes is changed for different scenarios. We compare the performance of the protocols in terms of their network utilization, and of their fairness. For simplicity, the simulations assume that a node knows the fate of any transmission

within the same time slot that it took place.

Network utilization Fraction of successful transmission, as defined in section 3.2.

Fairness We provide two measurements of fairness. The first is the *Jain's index*, as defined in section 3.2

The other measure we use is the *bottom-10% fair share.* To compute it, sort the nodes in order of bandwidth, so that $b_1 \leq b_2 \leq \cdots \leq b_n$, and let $m = \lceil n/10 \rceil$. Then, $B_{10} = \sum_{i=1}^{m} b_i$ is the cumulative bandwidth of the bottom 10% of the nodes, and

$$F_{10\%} = \frac{nB_{10}}{mB}$$

is the ratio between the actual bandwidth for the bottom 10%, and the bandwidth the bottom 10% would receive under fair allocation. The $F_{10\%}$ measure is a number between 0 and 1, just like Jain's index. While Jain's index captures the fairness of the overall allocation, the $F_{10\%}$ measure captures how the most "unfortunate" nodes fare in the protocol.

4.4.1 ALOHA-Q and ALOHA With Exponential Back-off

ALOHA-Q We implemented ALOHA-Q, the Q-learning version of slotted ALOHA proposed in [10, 9], using a frame length of $M = 64$. Each node stores *q-values* $q_1, q_2, \ldots, q_{64}$, where q_i represents the quality of the decision of transmitting in the i-th slot of the frame. In every frame, a node transmits in the slot i with maximal q_i;

if the transmission is successful, it increases q_i; if a collision occurs, it decreases q_i and it follows a randomized back-off before retrying. As long as the number m of active nodes is no larger than $M = 64$, the performance of ALOHA-Q converges to m/M in the long run; when $m > M$, conflicts for the use of the time-slots in the fixed-length frames arise, and the performance degrades. In our simulations, the number of active nodes is at most about 50, so as to use ALOHA-Q close to its optimum performance.

Slotted 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 $p := \alpha p$ in case of collisions, and $p := \min(1, p/\alpha)$ in case of empty slots, where α is a constant that determines adaptation speed; in our simulations we use $\alpha = 0.9$.

This is the *symmetrical* version of ALOHA-EB, in which all nodes transmit with similar probability. For large numbers of nodes, the bandwidth utilization reaches the optimal value of $1/e$, or about 37% [24]. In another version of ALOHA with exponential back-off, each node adjusts its transmission probability as a function of the success, or failure (collision), of its own transmissions only. For this "individual" version of ALOHA with exponential-backoff, it is known that one node will soon dominate and transmit all the time, while the other nodes reduce their transmission probability indefinitely. The network utilization approaches 100%, but the bandwidth is used by one node only. We do not provide comparison graphs for the "individual" version of

ALOHA with exponential backoff, as its behavior is well known, and as we are interested in protocols that allow nodes to share the bandwidth.

4.4.2 Simulation Results

4.4.2.1 Network With 50 Active Nodes

In Figure 4.2 we compare the network utilization and fairness of the protocols for a network consisting of 50 active nodes. As the protocols include randomization, we report the average and standard deviation computed over 25 simulations. We aggregate fairness over time blocks that contain on average 20 time slots for each network node: this ensures that the relative number of transmissions by the node are not unduly affected by statistical noise.

Both ALOHA-QT and ALOHA-QTF achieve a channel utilization above 75%. ALOHA-QT reaches 75% utilization in about 500 time slots (that is, in only 10 time-slots per node); the ramp-up of ALOHA-QTF is somewhat slower, and 1000 time-slots (or 20 slots per node) are required to achieve 75% utilization. It is remarkable that 50 nodes can coordinate their transmissions with just a few transmissions per node. The slower ramp-up of ALOHA-QTF is due to the modified weight update (4.3), which makes the nodes slightly slower in exploiting available network slots. This is the price to pay for the fairness of ALOHA-QTF: the protocol exhibits the highest fairness of the protocols considered, guaranteeing 75% of the average node bandwidth even to the nodes in the 10% lowest bandwidth percentile.

Once the nodes have time to fully adapt, ALOHA-Q should lead to a utilization

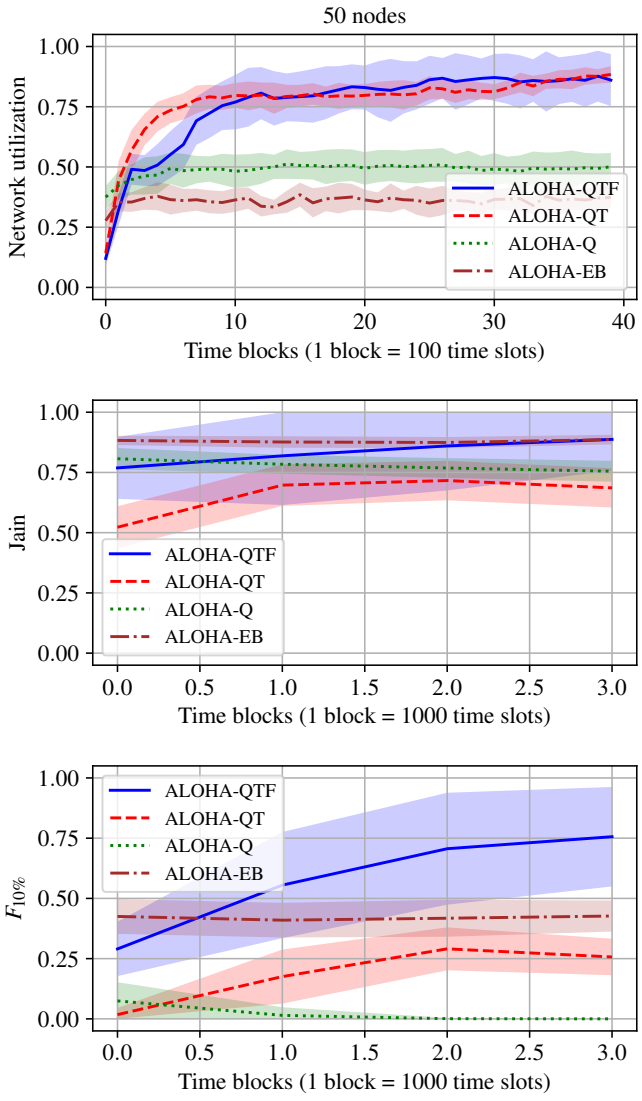


Figure 4.2: **Performance of ALOHA-QT and ALOHA-QTF: 50 Nodes.** Network utilization and fairness in a network with 50 active nodes. The results are the average of 20 simulations; the colored bands are plus or minus one standard deviation.

of $50/64 \approx 78\%$. However, in the 4,000 time-slots spanned by our simulation, the adaptation has not occurred. We note that ALOHA-Q has a $F_{10\%}$ fairness very close to zero, indicating that there is a group of nodes that consistently fails to be able to transmit successfully.

4.4.2.2 Variable Number of Active Nodes

To gain a better understanding of the performance of the protocols when nodes join and leave active transmission, we consider a scenario in which the number of active nodes is initially 20, then increases to 50, and finally decreases to 30, as indicated in Figure 4.3.

We give results for the average of 20 simulations for each protocol. We see that the bandwidth utilization of ALOHA-QTF is markedly superior to the utilization resulting under both ALOHA-Q and ALOHA-EB. The utilization of ALOHA-QTF declines only during and immediately after the ramp-down from 50 to 30 active nodes: as nodes become inactive, some time-slots are left empty, and the remaining nodes need some time to adapt and utilize these newly-available time-slots. ALOHA-EB is once again close to its optimal utilization of $1/e \approx 0.37$ throughout. The utilization of ALOHA-Q reaches 50% for 50 active nodes, again short of its theoretical limit of $50/64 \approx 78\%$.

The fairness of ALOHA-QTF dips temporarily when the number of active nodes rises from 20 to 50: the 30 newly active nodes need some time to gain a bandwidth comparable to the one of the 20 incumbents. In particular, the $F_{10\%}$ index for ALOHA-

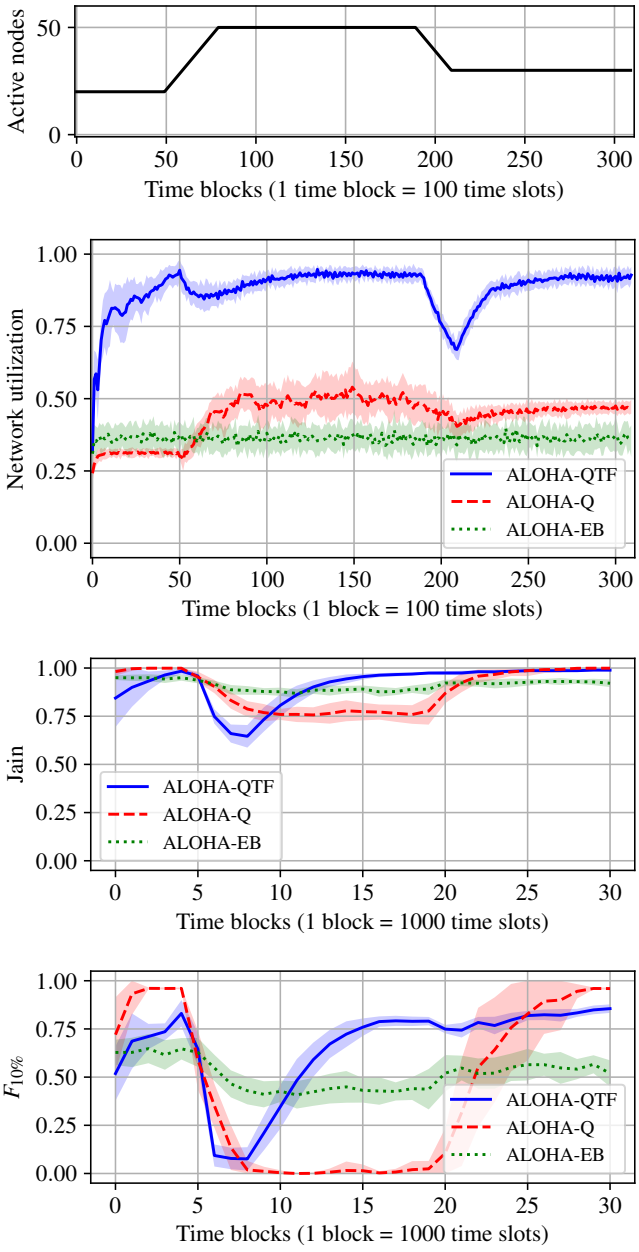


Figure 4.3: **Performance of ALOHA-QTF: Variable Number of Nodes.** Network utilization and fairness for ALOHA-QTF, ALOHA-Q, and ALOHA-EB under variable number of active nodes. The results are the average of 20 simulations; the colored bands are plus and minus one standard deviation.

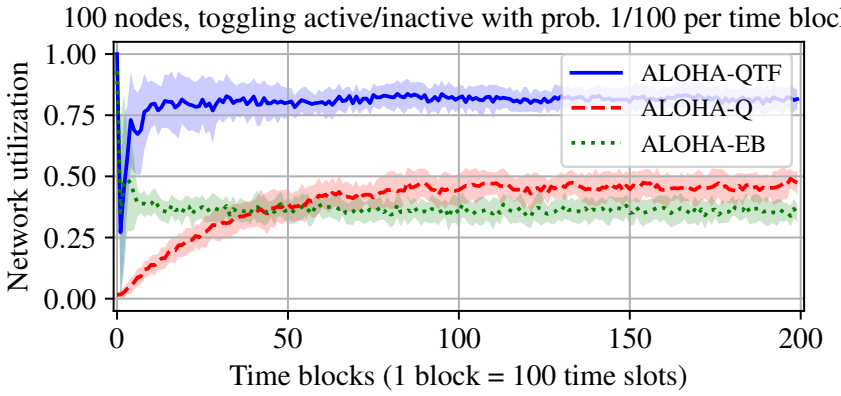


Figure 4.4: **Performance of ALOHA-QTF: nodes randomly joining and leaving.** Network utilization in a network of 100 nodes in which every node has probability 1/100 of toggling the active/inactive state every 100-time units block. The results are the average of 20 simulations; the colored bands are plus and minus one standard deviation.

FQT dips to close to 0 for a few thousand time-slots. The dip in $F_{10\%}$ is much more pronounced and long-lasting for ALOHA-Q.

4.4.2.3 Nodes Randomly Becoming Active and Inactive

Finally, we considered the case of nodes becoming active or turning inactive at random. We simulated a network with 100 nodes, of which only one is initially active. At each time block (where 1 time block = 100 time slots), each node has probability 1/100 of switching state, from inactive to active, or vice versa. Thus, on average, in each time block one node changes state. The utilization is reported in Figure 4.4. We see that ALOHA-QTF maintains its high level of utilization, above 75%, in spite of the nodes continually joining or leaving the set of active nodes. The utilization of ALOHA-Q grows with the number of active nodes, and then settles at about 50%. Again, the

utilization of ALOHA-EB is close to its 37% theoretical optimum.

4.5 Conclusion

We presented a new approach to the use of reinforcement learning in the context of slotted ALOHA that dramatically improves channel throughput. The proposed approach is based on the concept of policy trees, and strikes a balance between the two diametrically opposite approaches followed in ALOHA-Q and in the DRL-based approach. As in ALOHA-Q, the learning is based on a fixed set of schedules, carefully chosen to guide the nodes towards collaboration. As in the DRL-based approach of [45], there are no fixed transmission frames. This yields protocols that can quickly adapt to changing network conditions, achieving high and fair utilization under a wide range of number of active nodes and network traffic conditions, with none of the computational load required by training neural networks.

We presented two examples of the use of policy trees to access a shared time-slotted channel. The simpler ALOHA-QT is based on reinforcement learning applied to the policy tree that defines periodic schedules. Its refinement ALOHA-QTF modifies the policy tree weight update rules in order to improve the fairness of the bandwidth distribution among active nodes at the price of additional computation and storage requirements. These computational and storage costs are relatively modest, and consequently, ALOHA-QTF can be easily implemented on top of low-power embedded CPUs, or even in custom hardware. Simulation experiments illustrate the marked performance

improvements attained with ALOHA-QT and ALOHA-QTF compared to ALOHA-Q and slotted ALOHA with exponential back offs.

Chapter 5

Adaptive Tree ALOHA with Delayed Ack

5.1 APT-ALOHA

5.1.1 Overview: From AT-ALOHA to APT-ALOHA

APT-ALOHA[1] (Adaptive Policy Tree ALOHA) [48] is a protocol designed with AT-ALOHA [46] (Chapter 3) as a starting point. It's also used for nodes that share a time slotted transmission channel. In APT-ALOHA, the nodes also determine their transmissions by using a policy tree that's adaptive in its structure. However, the main difference between APT-ALOHA and AT-ALOHA is that AT-ALOHA nodes require knowing the channel outcome of every time slot immediately – what we call "immediate ack" in this thesis. In APT-ALOHA, it uses delayed acknowledgement

[1] APT-ALOHA is the result of collaboration with Marc Mosko, who designed and implemented the delayed ack mechanism in ns3. This is different from the other three protocols that are implemented in python. This chapter is included in this thesis for the sake of completeness but the credit goes to Marc.

mechanisms such as positive ack, negative ack and gossip protocol, so that nodes learn about their transmission outcome only after they receive transmission from their peers, and then deduct the fate of their previous transmissions. This allow APT-ALOHA to be used in ad-hoc networks where there is no central repeater to inform nodes of their transmission outcome immediately. APT-ALOHA uses the same policy tree (Figure 3.1) as AT-ALOHA, and uses very similar operations to update the policy tree. Therefore in this section we will refer to Chapter 3 for the when the concepts are the same, and elaborate on the parts that are different.

APT-ALOHA nodes can choose from periodic schedules with periods that are power of 2. A schedule (i, m) prescribes sending at all times t such that $t \mod 2^m = i$; we let $T(i, m) = \{t \mid t \mod 2^m = i\}$ be the set of times associated with schedule (i, m). A set of all such schedules $\mathcal{S} = \{(i, m) \mid m > 0, 0 \leq i < 2^m\}$ can be arranged in a tree with root $(0, 0)$. This is the policy tree (Figure 3.1). A *policy* for a node is a subset $\pi \subseteq \mathcal{S}$ of schedules, called the *active schedules*. The transmit times of a policy π are the union of the transmit times of the individual schedules in π, or $T(\pi) = \bigcup_{s \in \pi} T(s)$.

Nodes start with a simple policy tree consisting of a single active schedule, and evolve their policy tree in response to the empty slots, collisions, and packet acknowledgements they receive using the above operations. A node updates a policy tree by means of four operations:

- A *demotion* operation either removes a schedule from the set of active schedules, or replaces the schedule with a descendant in the tree. Demotion is used to help resolve conflicts: when a node determines that a collision occurred, it demotes the

responsible schedule.

- A *barge-in* operation adds a schedule to the set of active ones. When nodes detect an empty time slot, with a certain probability they do a barge-in, in an attempt to make use of future periodic occurrences of the time slot.

- A *normalization* operation ensures the minimal and canonical representation of the set of active schedules.

- A *pruning* operation ensures that every policy, after normalization, contains at most a fixed number of schedules. This bounds the computation and memory requirements of the protocol.

Compared to four operations that were introduced in AT-ALOHA [46] (Chapter 3); APT-ALOHA has refined definitions of demotion and barge-in to allow for a faster adaptation, necessary due to the delayed feedback provided by delayed acknowledgements.

Demotion The procedure $demote(\pi, t, k)$ is illustrated in Figure 5.1. If $t \notin T(\pi)$, then $demote(\pi, t, k)$ leaves π unchanged. If $t \in T(\pi)$, the procedures removes from π the unique schedule $(i, m) \in \pi$ such that $t \in T(i, m)$. Further, if $\{(i', m') \in \pi \mid m' < m\} = \emptyset$, that is, if the removed schedule was at minimal distance from the tree root, then the procedure will add a descendant schedule to the policy as follows. Let

$$k' = \max\{m + 1, k\}, \qquad (i', m') := (i, m),$$

and repeatedly pick

$$(i', m') := pick\{(i', m' + 1), (i' + 2^{m'}, m' + 1)\}$$

until $m' \geq k'$, then add (i', m') to π. The random choice of descendant helps nodes settle on non-conflicting policies; the level k to which demotion proceeds is discussed in Section 5.1.5.

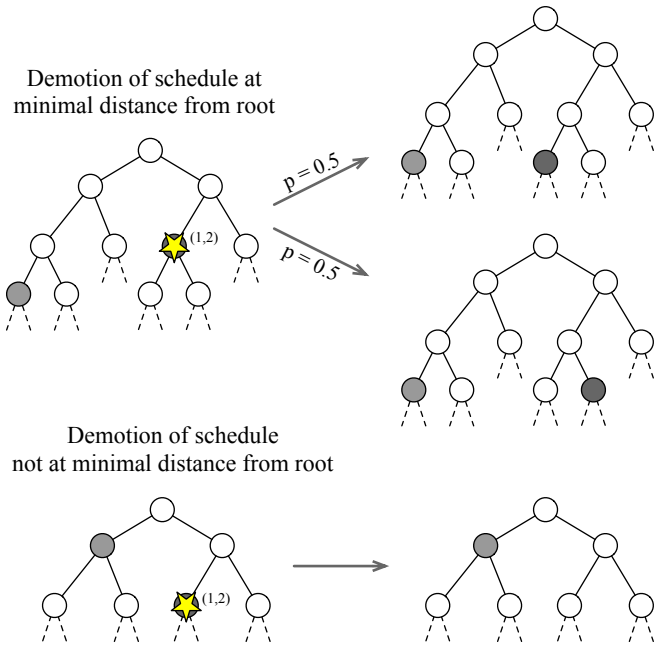


Figure 5.1: **APT-ALOHA Policy Tree Update - Demote**. Two cases of of demoting the starred schedule $(1, 2)$ in the policy tree via $demote(\pi, 1, 3)$.

Barge-in When a node detects that a time slot at time t is empty, it may add to the policy a schedule that causes it to transmit at future periodic occurrences of the slot. Precisely, $bargein(\pi, t, m)$ adds to π the schedule (i, m) with $t \mod 2^m = i$. The

added schedule will cause the node to transmit at $t + 2^m, t + 2 \cdot 2^m, t + 3 \cdot 2^m$, and so forth. Figure 5.2 illustrates which schedules are added by $bargein(\pi, 25, m)$, for different values of m. Barge-ins are not performed deterministically: doing so would cause many collisions, as all network nodes would try to exploit the same transmission opportunities. The probability of doing a barge-in, and the schedule insertion level m, are tuned by the protocol as detailed in Section 5.1.5.1.

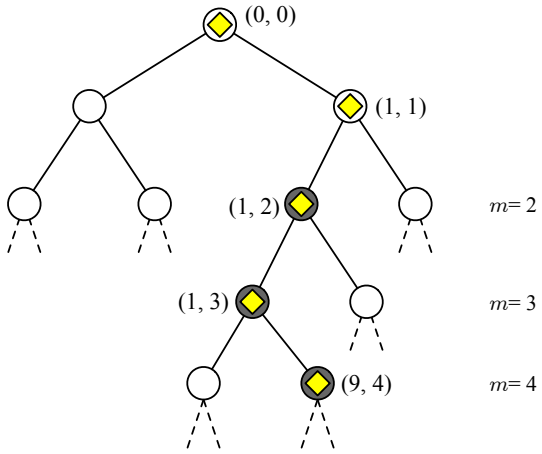


Figure 5.2: **APT-ALOHA Policy Tree Update - Barge In**. New schedule get added to a policy tree via barge-in procedure. The diamond nodes show the branch of the tree that would have contributed to transmission at a certain time slot, and the greyed nodes show the potential candidate schedules to be added

Policy normalization The normalization operation $normalize(\pi)$ is the same as in AT-ALOHA, described in Chapter 3, Section 3.1.4, Figure 3.4. Descendant of active schedules are eliminated, sibling schedules are merged.

Policy pruning After the policy is normalized, the policy is *pruned* to enforce a maximum number of active schedules, by limiting the policy depth in the tree, then the

number of active schedules in it. This is also the same as AT-ALOHA, described in Chapter 3, Section 3.1.4, Figure 3.5.

In the protocol implementation for which we will provide experimental results, after each *demote* and *bargein* operation, we normalize and prune the policy according to $\Delta = 2$ and $M = 10$, thus setting $\pi := \mathit{prune}(\mathit{normalize}(\pi), 2, 10)$.

5.1.2 The Algorithm

The APT algorithm is presented schematically in Algorithm 3; we describe below its structure and behavior.

5.1.3 Protocol Structure

State variables The state variables of an APT node include the time slot counter t, the policy $\pi \in \mathcal{P}$ and its labeling ℓ tracking which schedules have caused transmissions, the list A_{out} of outgoing acknowledgements, and the list $A_{pending}$ of pending acknowledgements along with their expiration times η. Acknowledgements and node labeling ℓ are described subsequently. A node maintains an estimate $\hat{N}$ of the number of other active network nodes, and a *kindness* probability p_k, tuned as described in Sec 5.1.5.

These state variables are local to each node; in particular, nodes do not need to agree on the numbering of time slots. A policy π associated with a time slot counter t is equivalent to a policy $\text{shift}(\pi, \Delta) = \{((i + \Delta) \mod 2^m, m) \mid (i, m) \in \pi\}$ associated with counter $t + \Delta$, both in its transmission times, and in its update behavior. Thus, nodes can simply start counting slot when they join the protocol. Because APT uses a

Constants:

s: ID of local node where APT is running;
$\alpha = 0.98$: kindness inertia;
$\beta = 0.05$: target fraction of empty slots;
$q_k = 10^{-2}$: kindness probability lower bound;
$\Delta_{new} = 2$: schedule insertion delta;

State Variables:

active: True if the node is active; false otherwise;
t: time slot counter;
π: APT policy, with schedule labeling ℓ;
$A_{pending}, A_{out}$: lists of pending and outgoing acknowledgements;
η: expiry time for acknowledgements;
$\hat{N}$: estimated number of active network nodes (see Sec 5.1.5.1);
p_k: kindness probability (see Sec 5.1.5.2);

Channel Variables:

$h \in \{S, E, C, R\}$: channel state at the end of a time slot.

Initialization:

$t := 0$; $p_b := 0.1$; $\pi := \mathit{choice}\{(0,1),(1,1)\}$;

At the beginning of each timeslot:

$h :=$ *channel outcome of previous slot in* $\{S, E, C, R\}$;
if $h = E$ **then**
  $p_k := \min(0.5, p_k \cdot \alpha^{1/\beta})$;
  with probability $1/\hat{N}$: $\pi = \mathit{bargein}(\pi, t, \lfloor \log_2(\hat{N} - 1) \rfloor)$
else
  $p_k := \max(q_k, p_k/\alpha)$
foreach $t' \in A_{pending}$ **do** (ack expiration)
  if $t - t' > \eta(t')$ **then**
    $\mathit{ldemote}(\pi, t', \lceil \log_2 \hat{N} \rceil)$;
    $A_{pending} := A_{pending} \setminus \{t'\}$
$\pi := \mathit{prune}(\mathit{normalize}(\pi), 2, 10)$;
$t := t + 1$;
if *there is* $(i, m) \in \pi$ *with* $t \mod 2^m = i$ **then** (transmit)
  remove from A_{out} a set A of acknowledgements in FIFO order;
  send a packet with acknowledgements A;
  $A_{pending} := A_{pending} \cup \{t\}$;
  $\eta(t) := t + 2^m$;

Upon receiving packet from sender u **with acks** A**:**

$A_{out} := A_{out} \cup \{(u, t, \text{T})\}$;
foreach $(s', \Delta, b) \in A$ **do**
  $t' := t - \Delta$;
  if $s' = s \wedge t' \in A_{pending}$ **then** (own ack)
    $A_{pending} := A_{pending} \setminus \{t'\}$;
    if $b = \text{T}$ **then** (kindness)
      with probability p_k do $\mathit{ldemote}(\pi, t', \lceil \log_2 \hat{N} \rceil)$
    else (demotion due to collision)
      $\mathit{ldemote}(\pi, t', \lceil \log_2 \hat{N} \rceil)$
  if $s \neq s'$ **then** (ack for another node)
    if $t' \in A_{pending}$ **then** (virtual collision sender)
      $A_{pending} := A_{pending} \setminus \{t'\}$;
      $\mathit{ldemote}(\pi, t', \lceil \log_2 \hat{N} \rceil)$
    if $b = \text{T} \wedge \exists s''.s' \neq s'' \wedge (s'', t', \text{T}) \in A_{out}$ **then** (virtual collision receiver)
      $A_{out} := A_{out} \cup \{(s'', t', \text{F})\} \setminus \{(s'', t', \text{T})\}$
    if $b = \text{T} \wedge (s', t', b) \in A_{out}$ **then** (redundant ack)
      $A_{out} := A_{out} \setminus \{(s', t', b)\}$

Algorithm 3: APT-ALOHA Algorithm.

binary policy tree, a node may freely wrap its counter t at any sufficiently large power of two, such as a common 32-bit counter.

Time slot decisions, and transmission attribution At each time slot t, APT transmits if there is a schedule $(i, m) \in \pi$ such that $t \mod 2^m = i$, and waits otherwise. Furthermore, we need to mind a subtle interaction between schedule tree updates and delayed acknowledgements. Due to a demotion or other tree operation, a schedule (i, m) that caused transmission might have been replaced by another schedule (i', m'), with also $t \mod 2^{m'} = i'$ by the time the acknowledgement is known to have failed (due to either timeout or a negative acknowledgement). We do not want to demote schedules that were not the ones that caused the original transmission. To this end, we use a labeling ℓ tracking which schedules triggered transmissions. When a schedule (i, m) is added to the policy tree (by a demotion, barge-in, or normalization) we set $\ell(i, m) = \text{F}$; we set $\ell(i, m) = \text{T}$ when the schedule triggers a transmission. We also introduce a procedure *ldemote*, such that $demote(\pi, t, k)$ modifies π only if both conditions apply:

- there is $(i, m) \in \pi$ with $t \mod 2^m = i$ (as in normal demotion), and
- $\ell(i, m) = \text{T}$.

This transmission attribution and modified demotion are necessary in presence of delayed acknowledgements.

Time slot status At the end of each time slot, the node's radio communicates to the protocol the status $h \in \{D, E, C, R\}$ of the time slot, where:

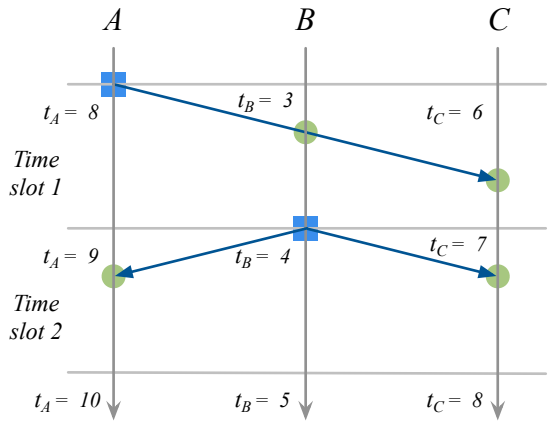
- S (send) indicates that the node transmitted a packet.
- E (empty) indicates that the time slot was empty: no energy above a certain threshold was detected, indicating that no node transmitted.
- C (collision) indicates that energy was detected, but no packet could be decoded.
- R (reception) indicates that the radio correctly decoded a packet during the time slot.

Acknowledgements In order to determine which packets are received correctly, APT-ALOHA relies on positive acknowledgements (ACKs) and negative acknowledgements (NAKs). Every node maintains two acknowledgement queues:

- the *pending acknowledgements queue* $A_{pending}$, which stores the timestamps of the node's transmitted, and thus far unacknowledged packets;
- the *outgoing acknowledgements queue* A_{out}, which stores the acknowledgements for received packets the node is waiting to transmit. These acknowledgements are stored in the format (s, t, b), where s is the ID of the node whose packet is acknowledged, t is the local node time of the packet being acknowledged, and $b \in \{\text{T}, \text{F}\}$ is the Boolean value of the acknowledgement, which is $b = \text{T}$ for an ACK and $b = \text{F}$ for a NAK.

When a node transmits at time t due to schedule (i, m), it inserts t in the pending acknowledgements queue $A_{pending}$, and sets $\eta(t) = t + 2^m$ as the expiry time of such acknowledgement. In this fashion, the expiry time of acknowledgements is

dependent on the periodicity of the schedules, and automatically adjusts to the number of active nodes.



Time Slot	Node	$A_{pending}$	A_{out}	Sent acks
1	A	$\{8\}$	$\emptyset$	$\emptyset$
	B	$\emptyset$	$\{(A, 3, \text{T})\}$	
	C	$\emptyset$	$\{(A, 6, \text{T})\}$	
2	A	$\emptyset$	$\{(B, 9, \text{T})\}$	
	B	$\{4\}$	$\emptyset$	$\{(A, -1, \text{T})\}$
	C	$\emptyset$	$\{(B, 7, \text{T})\}$	

Figure 5.3: **Packet acknowledgements in APT-ALOHA**. The table states the state of the A_{out} and $A_{pending}$ queues at the end of each time slot, along with any acknowledgements sent. Squares denote transmission events, and circles reception events.

When an outgoing acknowledgement (s, t, b) is sent over the channel at sender time $t' > t$, it is re-encoded as $(s, t' - t, b)$, so that in the channel acknowledgement times are expressed as difference from the current time. When a node receives an acknowledgement (s, Δ, b) at time t'', the acknowledgement is translated back into $(s, t'' - \Delta, b)$ before processing, and is thus translated back into the local time of the receiving node.

Acknowledgements are sent using a *gossip protocol:* if a node hears an acknowledgement coming from another node, it removes the acknowledgement if it is also

present in its own queue, as the corresponding packet has already been acknowledged. In our implementation, in order to guarantee an upper bound to the transmission length, we include in each transmission N_{acks} acknowledgements, adding then as many as can fit if the packet to be transmitted is short; we select which acknowledgements to send on an older-first basis. In our simulations we use $N_{acks} = 2$.

The acknowledgement mechanism is illustrated in Figure 5.3. In the first time slot, A sends a packet, which is received at B at local time 3, and at C at local time 6. Acknowledgements for this packet are stored in the outgoing queues of nodes B and C. In the second time slot, B sends a packet, and adds to it an acknowledgement for A's previous packet: the $(A, 3, \text{T})$ in B's outgoing queue is transmitted in relative time as $(A, -1, \text{T})$. This packet is received at A at local time 9, and at C at local time 7. Note how node C drops its outgoing acknowledgement $(A, 6, \text{T})$ when it hears $(A, -1, \text{T})$ from B: as acknowledgements are a gossip protocol, C no longer needs to acknowledge A's packet in time slot 1.

5.1.4 APT-ALOHA Events

The APT-ALOHA protocol responds to two events: the *time-tick* event, which occurs at time slot boundaries, and the *packet reception* event, which occurs whenever a packet is correctly received and decoded, and passed to the protocol.

5.1.4.1 Time-tick event

When the time-tick event occurs, the protocol first finalizes the time slot that just completed, and then decides whether to send a packer or wait.

Time slot finalization When finalizing the time slot t, the node examines the outcome $h \in \{S, E, C, R\}$, and handles $h = E$ and $h = C$ as follows:

- $h = E$ (empty): the node with probability p_b executes *bargein*(π, t, κ); the choice of the insertion level κ will be detailed in Subsection 5.1.5.1.
- $h = C$ (collision): the node executes *ldemote*$(\pi, t, \lceil \log_2 \hat{N} \rceil)$, where $\hat{N}$ is an estimate of the number of active nodes, obtained as in Sec 5.1.5.1.

Next, the node checks the acknowledgements in the pending queue $A_{pending}$. If $t' \in A_{pending}$ and $t > \eta(t')$, the packet sent at t' is considered lost: t' is removed from $A_{pending}$, and the node executes *ldemote*$(\pi, t', \lceil \log_2 \hat{N} \rceil)$ to demote the schedule that caused the lost transmission. The policy π is then normalized and pruned, and time-counter t is incremented.

Send decision The node transmits if $t \in T(\pi)$ and it has a data packet ready, and waits otherwise. If the node transmits, it adds t to $A_{pending}$, and sets $\eta(t) = t + 2^m$, where (i, m) is the schedule in π that caused the transmission. If a packet is sent, the node dequeues in FIFO order due acknowledgements in A_{out}, translates them into transmission format, and adds them to the packet. In our implementation, a minimum of two such acknowledgements can be fit into a transmission.

5.1.4.2 Packet reception event

A packet, as received from the network, consists of a sender ID, a destination ID, a list of acknowledgements, and a message. If the destination ID is equal to the ID s of the receiving node, the message is passed to the node for processing. All acknowledgements $a = (s', t', b)$ received are processed as follows:

- *Virtual collision sender.* If $s \neq s'$, but $t' \in A_{pending}$, this means that a node is acknowledging a packet sent at the same time in which the node s also sent a packet. The node s can infer that a collision occurred, and it executes $ldemote(\pi, t', \lceil \log_2 \hat{N} \rceil)$.

- *Virtual collision receiver.* If $b = \text{T}$ and there is $(s'', t', \text{T}) \in A_{out}$ with $s'' \neq s'$, we have two acknowledgements for two different senders, both at time t'. This is an indication of a collision, and thus, we replace $(s'', t', \text{T}) \in A_{out}$ with (s'', t', F).

- *Removal of redundant acknowledgements.* If $b = \text{T}$, the acknowledgement (s', t', T), if found in $A_{pending}$, is removed, as the packet has already been acknowledged. If $b = \text{F}$, remove any $(s'', t'', b'') \in A_{pending}$ where $t'' = t'$. Thus, a NAK erases both ACKs and NAKs for the same time slot from $A_{pending}$.

- *Acknowledgements of own packets.* If $s' = s$, and $t' \in A_{out}$, then t' is removed from A_{out}. There are then two cases, for positive and negative acknowledgements.

  - If $b = \text{T}$, with probability p_k, the node executes $ldemote(\pi, t', \lceil \log_2 \hat{N} \rceil)$. Thus, after a successful transmission, a node relinquishes use of the time slot with small probability. This ensures that even in the absence of colli-

sions, time slots are not forever allocated to the same node, helping to make the protocol fair in the long term.

– If $b = \text{F}$, the node accounts for the collision by executing $ldemote(\pi, t', \lceil \log_2 \hat{N} \rceil)$.

5.1.5 Neighborhood Size and Kindness Probabilities

The estimated number of network nodes $\hat{N}$, and the *kindness probability* p_k of relinquishing a schedule, are dynamically tuned as follows.

5.1.5.1 Neighborhood size, demotion level, and barge-in probability

Every APT node computes an estimate $\hat{N}$ of the number of active network nodes by observing how many distinct node IDs it receives as part of messages or acknowledgements in recent time slots. For this purpose, the node collects pairs (s, t) of sender IDs s and times t into a list I, built as follows:

- in a message, s is the sender id, and t is the message's time;
- in an acknowledgement, s is the id of the message being acknowledged, and t is the time of the message being acknowledged.

The node then computes an estimate $\hat{N}$ of the neighborhood as the number of IDs that have been seen in the last L times:

$$\hat{N} = 1 + \left|\{s \mid \exists (s, t') \in I \wedge t - t' \leq L\}\right| ,$$

where t is the current time, and the 1 accounts for the node itself.

Once the estimate $\hat{N}$ is available, we choose for the "barge-in" probability $p_b = 1/\hat{N}$: if all the $\hat{N}$ nodes transmitted at the same time with probability p, the value of p that maximizes success (one, and only one, node transmitting) is $p = 1/\hat{N}$. The new policy is added at level $\kappa = \lfloor \log_2(\hat{N} - 1) \rfloor$, to ensure that its period is sufficient to accommodate existing transmissions. This level κ is also used for the demotion operation. For the length of the observation window, we choose $L = 3\kappa$, ensuring the window is large enough to observe most nodes.

5.1.5.2 Tuning the kindness probability

The kindness probability p_k is tuned so that a prescribed fraction β of time slots are left free. This ensures fairness, since it forces nodes to relinquish their time-slots with non-zero probability, and the free time slots can be claimed by any node via a barge-in. In our implementation, we use $\beta = 0.05$, striking a balance between high network utilization, and fairness. Initially, when a node becomes active, we arbitrarily set $p_k = 0.05$. Thereupon, nodes update p_k according to the channel outcomes E and C, R, D, as follows:

$$E: \ p_k := p_k \cdot \alpha^{1/\beta} \qquad C, R, D: \ p_k := p_k/\alpha$$

where $\alpha_k = 0.98$ is a coefficient determining the adaptation speed. Thus, the value of p_k decreases whenever there is an empty slot, and increases otherwise. The value stabilizes when the fraction of empty slots is β. The choice of $\alpha = 0.98$ leads to an adaptation

time of the order of $1/\alpha = 50$ transmission slots. The use of the lower bound q_k for p_k stabilizes the algorithm.

5.2 Performance Evaluation

We simulate the performance of APT-ALOHA in the NS3 simulator [34]. Our scenario consists of a peer-to-peer single-hop network using a long-range PHY with a time-slotted channel. This is similar to some tactical waveforms, though these results could apply to some long-range commercial systems such as Wi-Fi HaLow (802.11ah) or the LoRa LPWAN, which uses ALOHA.

We use a frequency-hopping spread-spectrum physical layer operating around 1 GHz. The time-slot length is approximately 7 msec, and the maximum MAC payload is 52 bytes per slot. The PHY data rate is just under 64kbps. The PHY, as simulated, has a maximum transmission distance of over 50Km. In each simulation run, nodes are uniformly distributed random in a 50 Km by 50 Km grid, which has just under a 167 μsec maximum propagation time. Nodes transmit at 50 dBm, and the average receive power of decoded packets is -68 dBm with a standard deviation of 6.28 dBm. In receive mode, if APT-ALOHA or ALOHA-EB do not decode a packet, they use a noise threshold of -80 dBm to determine if a slot is occupied based on energy detect. In such a case, they label the slot as a collision instead of empty.

A key feature of APT-ALOHA is the minimal information exchanged between nodes. The only required signaling is the acknowledgements (ACK). Neighbor discovery

rides on existing transmitter source IDs and source IDs in ACKs. In our experiments, the average wait time to receive an ACK is under 1.2 slots (stdev 0.08 slots) at steady state. The maximum wait time we observed was 19 slots.

Comparable low-power wide-area networks (LPWANs) are SigFix, LoRaWAN, NB-IoT, and Wi-Fi HaLow (802.11ah) [27, 2]. These protocols have maximum payloads of 12 bytes (SigFox), 243 bytes (LoraWAN), 1600 bytes (NB-IoT), and 100 bytes (802.11ah). They are intended for use over ranges from 1km (802.11ah) to 40km (SigFox rural). NB-IoT and 802.11ah both rely on base stations or access points to coordinate communications. LoRaWAN class A devices use ALOHA channel access with acknowledgement. Our evaluation indicates that LoRaWAN is a prime candidate for improved channel access by adopting APT-ALOHA and should be evaluated in future work.

We compare APT-ALOHA with ALOHA-EB with delayed ACKs [22], and with AT-ALOHA [46], ALOHA-QT [13], and ALOHA-Q [10, 9] relying in *immediate* ACKs.

In ALOHA-EB, a node updates its transmission probability $p(t)$ at every slot and transmits in a slot with probability $p(t)$. ALOHA-EB assumes instantaneous knowledge of the outcome of each transmission. For comparison purposes, ALOHA-EB and APT-ALOHA must have the same ability to sense outcomes. Accordingly, we modified ALOHA-EB to use the same slot outcome detection mechanism as APT-ALOHA, with ACK messages sent in subsequent time slots. In our implementation, ALOHA-EB uses the same PHY layer as APT-ALOHA with the same PDU size and slot size, and the same ACK data structure, so the protocol overhead from each ACK is the same.

ALOHA-EB does not use the APT-ALOHA NACK mechanism.

5.2.1 Metrics and Simulation Runs

In our simulations, each node is always backlogged with traffic to send. This models the most difficult situation for ALOHA, when the channel demand is $G = 1$. For each time slot, we observe the radio channel to determine if there was 0, 1, or more than one transmission. If there was 0, we declare the slot Empty; tf there was 1, we declare the slot a Success; and if there was more than one transmission, we declare the slot a Collision. We measure the fraction of slots that are empty, success, or collisions; in particular, the network utilization is the fraction of slots that are declared Success. We note that in some slots, it is possible that a node is able to capture a packet even though multiple packets were sent; thus, each node may measure more successes than the above, network-wide, statistics.

We use network utilization and jain index to measure the performance and fairness of of the protocols, as defined in section 3.2.

To show both the fast initial adaptation of the protocol, and the subsequent adaptation to network changes, we ran a simulation that begins with 10 nodes, ramps up to 50 nodes, then ramps down to 30 nodes. The simulation is repeated for 10 trials with different random seed; our figures display the average and sample standard deviation of the measured metrics.

5.2.2 APT-ALOHA Compared with ALOHA-EB

Fig. 5.4 shows the simulation scenario, which is made up of 5 segments, as shown in Fig. 5.4(d). 10 nodes are active and then 40 nodes begin joining one per block. Then all 50 nodes are then active until the first 20 begin deactivating one per block, leaving the remaining 30 nodes stay active to the end of the simulation.

Table 5.1 summarizes the protocol performance per segment. The numbers reported in the table are the average rate for each metric during that period. The ALOHA-EB success rate during the steady-state periods is between 12.7% and 26.3%. These are very good numbers for a slotted ALOHA protocol under constant channel demand, as the maximum performance of traditional slotted ALOHA is 36.8% under optimal load and near 0% for 10 or more nodes always ready (immediate first transmission). The bulk of the channel time (61.3% - 83.0%) is spent in the collision state, even during the steady-state periods.

Segment	Success		Collision		Empty	
	APT	EB	APT	EB	APT	EB
10 nodes	83.8%	26.3%	2.1%	61.3%	14.1%	12.4%
ramp up	69.9%	16.4%	12.0%	77.3%	18.1%	6.3%
50 nodes	88.1%	12.7%	3.0%	83.0%	8.9%	4.3%
ramp dn	67.9%	12.7%	8.4%	82.9%	23.7%	4.4%
30 nodes	87.2%	15.9%	4.3%	78.0%	8.5%	6.1%

Table 5.1: Average network utilization during different phases of the ramp simulation, for APT-ALOHA and ALOHA-EB.

In contrast, the APT-ALOHA success rate during the steady-state periods is between 83.8% and 87.2%. During the ramp up segment, when 30 nodes are added, the

APT-ALOHA success rate dips to 69.9% with a corresponding increase in the collision rate to 12.0% as nodes learn new non-conflicting schedules. During the ramp down segment, the APT-ALOHA success rate dips to 67.9% with a corresponding rise in the empty rate as the neighborhood sizes adjust down to 30 nodes. There is a slight rise in the collision rate as nodes probe for new non-conflicting schedules. Fig. 5.4c shows the Jain's Fairness index for the two protocols.

5.2.3 APT-ALOHA Compared to AT-ALOHA and ALOHA-QT

As we mentioned in our review of prior work, AT-ALOHA and ALOHA-QT depend on the ability of nodes to determine the success of transmissions through the intervention of a central node and two orthogonal channel. By contrast, APT-ALOHA with can work on single-channel ad-hoc networks. Therefore, a direct comparison APT-ALOHA with AT-ALOHA and ALOHA-QT is not possible. However, we offer a comparison in which the protocols were subjected to the same network dynamics (number of active nodes through time, and transmission load) used for Figure 5.4. APT-ALOHA was simulated in our described scenario, whereas AT-ALOHA and ALOHA-QT were simulated in an ideal network with immediate acknowledgements, no capture, no signal-to-noise ratio problems, and no propagation issues.

The results are presented in Figure 5.5. As we can see, the network utilization achieved by APT-ALOHA is very close to the one of AT-ALOHA and ALOHA-QT, indicating that our adaptation scheme is effective in presence of delayed acknowledgements. The main difference consists in a temporarily lower network utilization when the num-

ber of active nodes decreases from 50 to 30; APT-ALOHA is apparently slightly slower in exploiting newly available empty slots, compared to AT-ALOHA and ALOHA-QT.

5.2.4 APT-ALOHA Compared to ALOHA-Q

Again, a direct comparison for APT-ALOHA and ALOHA-Q is not possible, because ALOHA-Q relies on immediate acknowledgements. Nevertheless, some general comparison is possible. The chief limitation of ALOHA-Q is its fixed frame-length: for a frame-length M, and n active nodes, the utilization of ALOHA-Q is bounded by n/M if $n < M$, and rapidly degrades to the one of ALOHA-EB for $n > M$. Thus, even with a repeater, ALOHA-Q is not able to achieve high utilization for all of $n = 10, 50, 30$, regardless of the chosen value for M.

The comparison results are provided in Figure 5.6. We used the same simulation scenario as for the previous cases, and used a frame length equal to the maximum number of active nodes in our simulation (i.e., 50) for ALOHA-Q. We observe that the performance of ALOHA-Q is inferior to APT-ALOHA, even when the number of active nodes is 50. Indeed, while in this case the theoretical utilization of APT-ALOHA approaches 1, from [10], an adaptation period of the order of hundreds of thousands of time slots would be needed to reach such utilization.

5.3 Conclusion

We introduced the Adaptive Policy Tree (APT) algorithm to quickly approach collision-free transmissions and fairness over a common channel using the slotted

ALOHA protocol. In contrast to prior approaches that use machine learning to improve the performance of slotted ALOHA, the resulting protocol, APT-ALOHA, does not assume immediate acknowledgements, repeaters, or dedicated uplink and downlink channels, and does not require the definition of transmission frames with a fixed number of time slots per frame. Simulation results illustrate that APT-ALOHA attains far better throughput and fairness than slotted ALOHA with exponential backoff, and incurs only a fraction of the collision rate of ALOHA-EB. The performance of APT-ALOHA is close to that AT-ALOHA and ALOHA-QT, in spite of the reliance of these protocols on immediate acknowledgements.

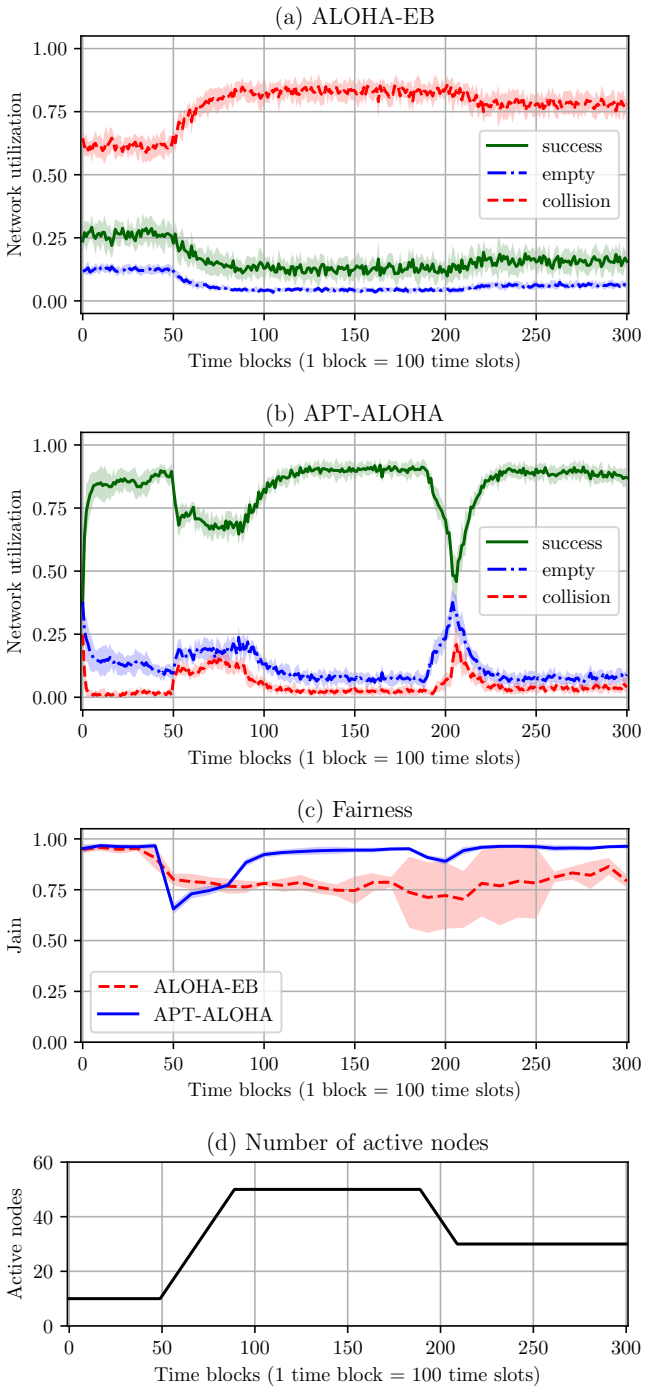


Figure 5.4: Network utilization in the ramp experiment for ALOHA-EB (a) and APT-ALOHA (b), and Jain's Fairness (c). The number of active nodes is shown in (d).

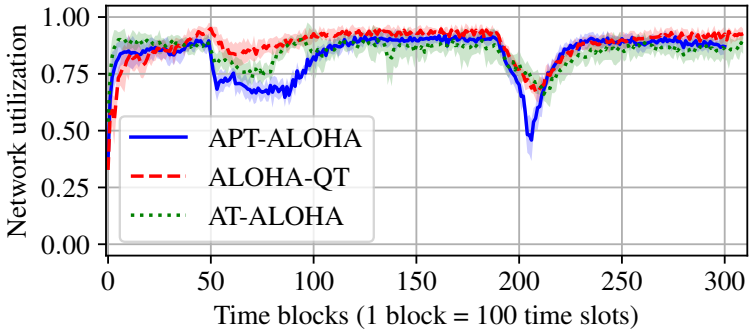


Figure 5.5: Network utilization of ramp experiment for APT-ALOHA, ALOHA-QT and AT-ALOHA performs similarly.

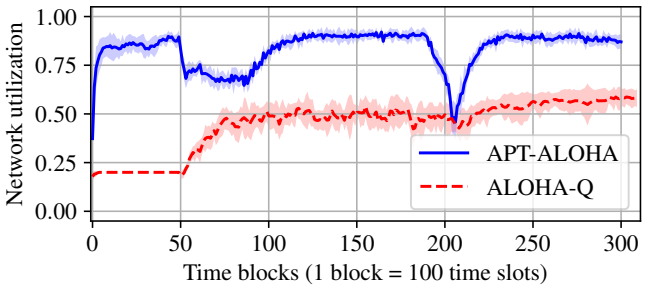


Figure 5.6: Network utilization comparison of APT-ALOHA and ALOHA-Q in ramp experiment

Chapter 6

Quantitative Tree ALOHA with Delayed Ack

6.1 Learning the Schedules

ALOHA-dQT [47], like its predecessor ALOHA-QTF [12] introduced in Chapter 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 a quantitative policy tree to allow the nodes to coordinate, and schedule their transmissions in a way that reduces collisions while allocating bandwidth fairly.

The 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 tree structure and the same operations on such structure, which are summarized in this section (Section 6.1). The acknowledgment structure and how the information drives reinforcement learning are presented in the following sections.

6.1.1 The Quantitative Policy Tree

Same as in ALOHA-QTF [12], 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 $\mathcal{P} = \{(i, 2^m) \mid 0 \leq i < 2^m, 0 \leq m \leq n\}$, 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 schedule, that is, its ability to prescribe transmitting without causing collisions.

The schedules can be organized into a tree, illustrated in Figure 4.1, where schedule (i, m) has as children $(i, m+1)$ and $(i+2^m, m+1)$. The schedules at the same tree level have the same period but different offsets, and thus prescribe non-colliding transmissions; every child schedule 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. We experimented using the (larger) set of schedules "transmit when $t \mod k = i$" for $0 \leq i < k \leq 2^n$. 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 times. 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.

6.1.2 The Policy Tree Updates

Policy-tree protocols, such as the ALOHA-QTF protocol [12] (Chapter 4) and the protocols presented in this chapter, consist in a *weight initialization step*, which is then followed by three steps performed cyclically:

1. *Schedule selection:* at the start of each network time-slot, a set of *active schedules* is selected; these schedules drive the decision to transmit, or to wait.

2. *Weight update:* at the conclusion of each network time-slot, the weights of all schedules are updated according to the outcome of the time slot.

3. *Weight normalization:* following the weight update, the weights are normalized: the weight from "losing" schedules is redistributed to other schedules, and the numerical values of the weights are normalized to ensure that they fall within a prescribed range.

We describe these steps in detail below.

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

$$w_{(i,m)} = \beta \cdot \frac{0.9 + 0.1 \cdot X_{(i,m)}}{1.2^m} \ , \tag{6.1}$$

where $\{X_\sigma\}_{\sigma\in\mathcal{P}}$ is a set of random variables independently sampled from the uniform distribution over $[0,1]$. In ALOHA-QT and ALOHA-QTF (Chapter 4) the value $\beta = 0.2$ was used. The denominator 1.2^m makes it so that nodes are initially likely to adopt schedules that transmit frequently, falling back on schedules that transmit more rarely only as needed to avoid collisions.

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

$$\mathcal{A}_t = \{\arg\max_\sigma w_\sigma\} \cup \{\sigma \mid w_\sigma \geq w_h\}\ , \tag{6.2}$$

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. In our implementations, we use $w_h = 0.95$. 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 $\sum_{\sigma\in\mathcal{E}_t}\delta(\sigma,t) > 0$, 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

$$w'_\sigma = w_\sigma \cdot \exp\big(\alpha\, X_\sigma^\gamma\, \delta(\sigma,t)\big)\ . \tag{6.3}$$

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 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 setting, schedules) become less effective [17, 18]. 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 weight given to each schedule, 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 schedules is bound to the $[q_{floor}, 1]$ interval, setting:

$$w_\sigma = \max(q_{floor}, \min(1, w_\sigma)) . \tag{6.4}$$

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

6.1.3 The ALOHA-QTF Protocol

The ALOHA-QTF protocol [13] 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:

$$\alpha = \begin{cases} 0.2 & \text{if } (d,h) \in \{(W,E),(T,S)\}; \\ -0.5 & \text{if } (d,h) \in \{(W,S),(W,C),(T,C)\}. \end{cases} \tag{6.5}$$

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 $w' := \mathit{normalize}(w, t)$ as described in the previous section, using a threshold $q_{\mathit{floor}} = 0$.

To this basic scheme are added two enhancements to ensure fairness (see [13] 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 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 (6.5) is performed not according to α, but according to α' given by:

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

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.

6.2 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 thus introduce an acknowledgement 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 acknowledgement 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 6.3 we will discuss in detail the changes in adaptation coefficients that best accommodate networks of nodes with, and without, energy detection.

6.2.1 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 [3, 11]. 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.
  - 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 6.1.

The rules in Table 6.1 can be understood as follows.

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 6.1: **ALOHA-dQT History Merging**. The merged historical state is indicated as function of the current history value h and incoming history value h'. Cells indicated with $*$ should not occur under normal protocol conditions.

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 6.1 reports the safest conclusion the protocol can draw.

	$\mathcal{H}^1@n_1$				$\mathcal{H}^2@n_2$			
t	$\mathcal{H}^1_3$	$\mathcal{H}^1_2$	$\mathcal{H}^1_1$	$\mathcal{H}^1_0$	$\mathcal{H}^2_3$	$\mathcal{H}^2_2$	$\mathcal{H}^2_1$	$\mathcal{H}^2_0$
6	W	s	W	**T**	W	s	W	**W**
7	s	W	**C**	$\bar{s}$	s	W	**W**	$\bar{T}$
8	W	**C**	$\bar{s}$	T	W	**c**	$\bar{S}$	s
9	**C**	$\bar{s}$	S	W	**c**	$\bar{S}$	s	W

Table 6.2: **ALOHA-dQT channel stated detection history merging** An example of collision detection and successful transmission acknowledgement for two nodes (n1 and n2) that do not detect energy. A 4-step history for these two nodes are shown, from $t = 6$ to $t = 9$. The left 4 columns represent channel history for $n1$ and the right 4 columns represent channel history for $n2$ The boldface and over-line symbols track the outcome of two 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 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 6.2 illustrates how the acknowledgment mechanism enables a node to detect that a collision occurred, for nodes that cannot detect channel energy. We depict only a 4-step history for two nodes n_1 and n_2; the nodes start at times $t = 6$ and end at $t = 9$. However, note the time counter doesn't have to be synchronized across nodes. Here history at the end of each time slot is shown, in every row. The left 4 columns represent channel history for $n1$ and the right 4 columns represent channel history for $n2$.

- At step $t = 6$, 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 $t = 7$, n_1 receives a packet from n_2, and marks s in $\mathcal{H}_0^1$. It then updates $\mathcal{H}_1^1 := r(T \triangleleft \mathcal{H}_1^2) = r(T \triangleleft W) = C$, because the W received from n_2 leads n_1 to believe that its previous transmission has caused a collision.

- At $t = 8$, n_1 transmits a packet, which is received by n_2; n_2 marks s for the most recent history, and it updates the T for its own previous transmission into a S using $\mathcal{H}_1^2 = r(T \triangleleft \mathcal{H}_1^1) = r(T \triangleleft s) = S$.

- At $t = 9$, 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.

6.2.2 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 according to the outcome of the time slot, and due to 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

$$w' = U(w, \alpha_i, t - i, \gamma_i) \tag{6.6}$$

where the multiplicative coefficients α_i, and the randomization amounts γ_i, are dependent on h_i and h_i'', and are specified in Table 6.3. In this multiplicative update equation (6.6), the time $t-i$ is the absolute time to which history position i refers. The coefficients in Table 6.3 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 reduce the weight of schedules 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 schedules that caused the collision storms.

- If the previous state T is updated to S, we successfully transmitted, and we deterministically increase the weight of schedules that caused that transmission.

- If the previous state W and it is updated to E, the slot is free, and we increase the weights of schedules that would make use of the slot, using randomization to break ties among nodes.

- If the state is updated to C, c, or s, regardless of whether we transmitted or not, it means that there is contention in the use of this slot, and we reduce the weights

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

Table 6.3: **Coefficients for multiplicative update**. The coefficients for update equation (6.3) differs according to the channel outcome; h refers to the channel outcome before the update, and h'' to the channel outcome after the update. In general, when $\alpha < 0$, schedule weights are reduced, to avoid collisions or potential collisions, and when $\alpha > 0$, schedule weights are increased, to claim empty slots.

of schedules 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 slightly increase the weight of the schedules 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 schedules that would have made use of the slot.

The update equation (6.6) are performed for all positions $1 \leq i \leq N$ of the history, after which the weights are re-normalized via $w' := \mathit{normalize}(w, t)$. From Table 6.3 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 6.3 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 4. We note that the algorithm uses the same fairness improvements as ALOHA-QTF, presented in Section 6.1.3.

6.3 Performance Evaluation

6.3.1 Protocols

We compare the performance of ALOHA-dQT with that of its direct predecessor, ALOHA-QTF [12], as well as with that of ALOHA-Q, the reinforcement-learning protocol proposed by [10, 9], 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 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 Chapter 4 and summarized in Section 6.1, as well as ALOHA-Q and ALOHA with exponential backoff (ALOHA-EB).

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 (6.1);

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 (6.1);

At every time slot:

`// Decision`
if $\sum_{\sigma \in \mathcal{E}_t} \delta(\sigma, t) > 0$ **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 (6.6) 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`
$w := \mathit{normalize}(w)$;
$t := t + 1$;

Algorithm 4: ALOHA-dQT Algorithm.

ALOHA-Q We implemented ALOHA-Q, the Q-learning version of slotted ALOHA proposed in [10, 9]. 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% [24].

6.3.2 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 relays 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 4. Protocol modules for ALOHA-EB, ALOHA-Q, and ALOHA-dQT-NE nodes can be similarly

implemented.

6.3.3 Performance Metrics

We evaluate protocols by measuring their network utilization and Jain index fairness, as defined in Section 3.2

6.3.4 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 6.1(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. Figure 6.2(c) shows the average number of active nodes throughout the simulation.

6.3.5 Results

Comparison With ALOHA-QTF, ALOHA-EB and ALOHA-Q The results for the ramp scenario are reported in Figure 6.1, and those for the churn scenario in Figure 6.2. We see from Figures 6.1(a) and 6.2(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

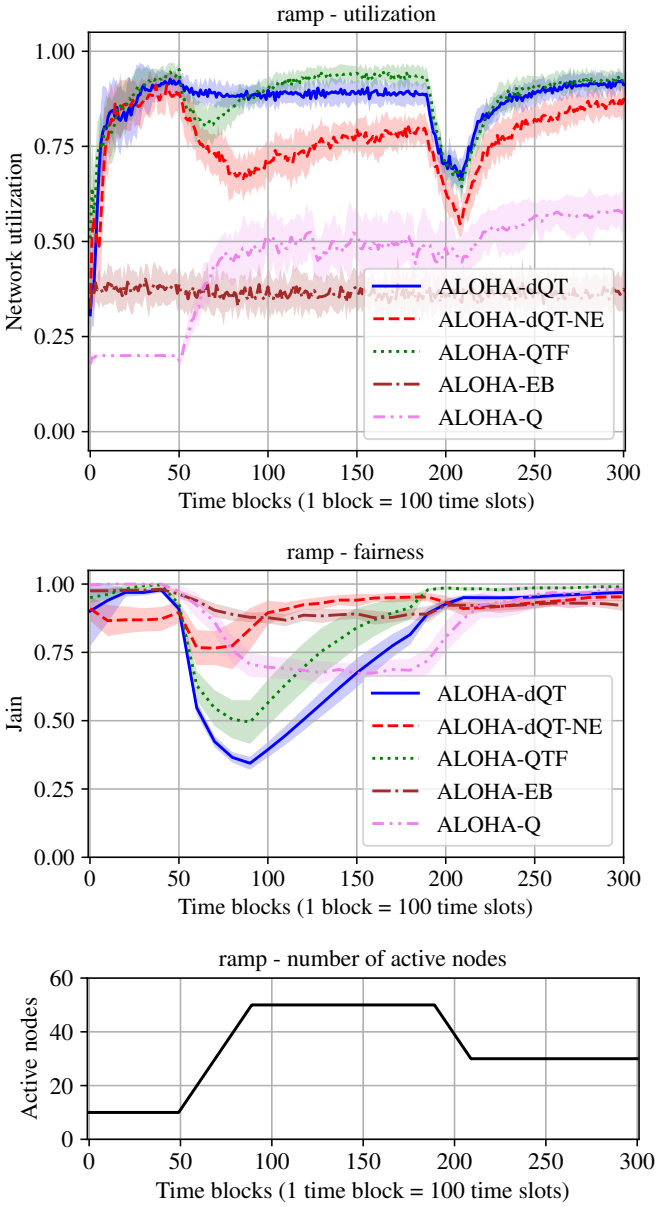


Figure 6.1: **ALOHA-dQT Ramp Exerpiment Result**. (a) utilization of five protocols. (b) fairness of five protocols. (c) number of active nodes through simulation. In (a) and (b), the solid lines are the average of 20 simulations; the colored bands are plus and minus one standard deviation.

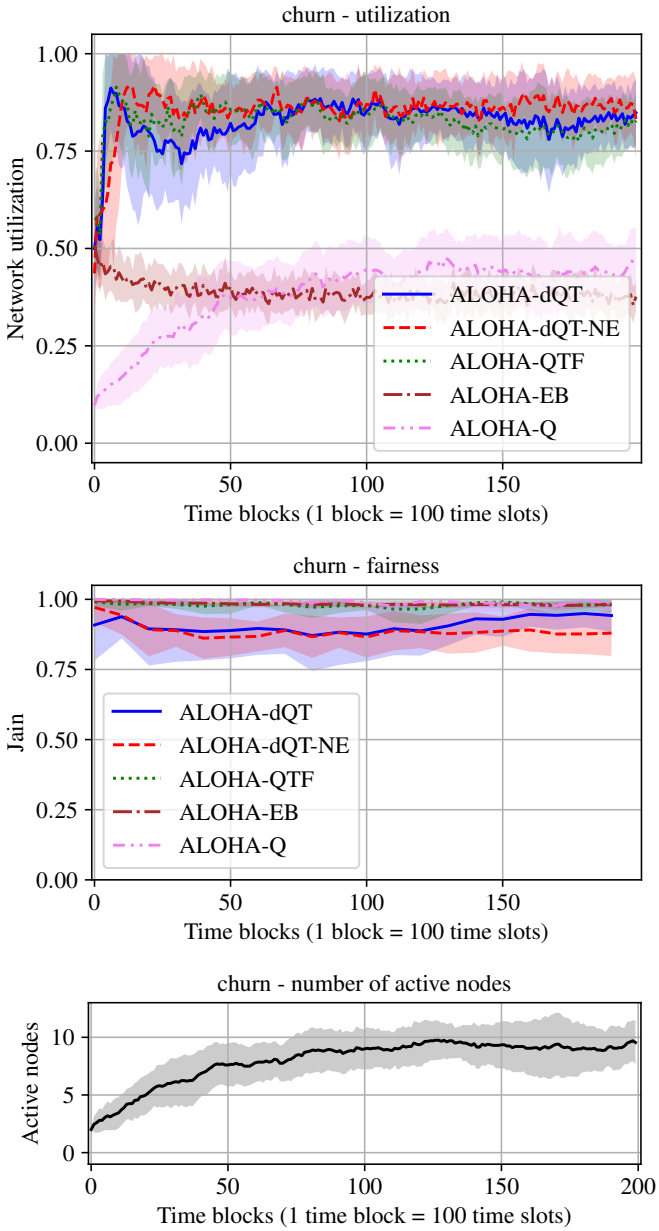


Figure 6.2: **ALOHA-dQT Churn Experiment Result.** (a) utilization of five protocols. (b) fairness of five protocols. (c) number of active nodes through simulation. In (a) and (b), the solid lines are the average of 20 simulations; the colored bands are plus and minus one standard deviation.

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 [10]. 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 6.1.3, and the next section will detail the effect of this fairness parameter and another parameter q_{floor}.

6.3.6 Hyper-parameter Analysis

The performance of the ALOHA-dQT protocol depends on several parameters, including the choice of the update coefficients of Table 6.3, the relinquish probability

ϵ_r of ceasing transmissions in a slot, and the weight floor q_{floor} for the interval $[q_{floor}, 1]$ of schedule weights. The update coefficients in Table 6.3 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 slots are not permanently held by the same nodes. Furthermore, for the reassignment of transmission slots to be effective, it is important that the schedule weights be bounded away from 0. To see this, consider what happens at node B when a schedule that was used in transmissions by node A is relinquished. At node B, several schedules would have caused transmission in the slot: precisely, all schedules (i, m) with $t \mod 2^m = i$. If the slot was in regular use by a periodic schedule of node A, the schedules of node B associated with the slot would have had a weight close to the weight floor q_{floor}, due to the negative weight 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 schedule weights to climb from q_{floor} all the way to w_h which is the threshhold for schedule selection (see (6.2)). 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 two parameters depend on the node's ability to detect slot energy; we discuss the two cases separately.

ALOHA-dQT In Figure 6.3 we depict the effect of varying the weight 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.

Figure 6.4 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 $\epsilon_r = 0.02$.

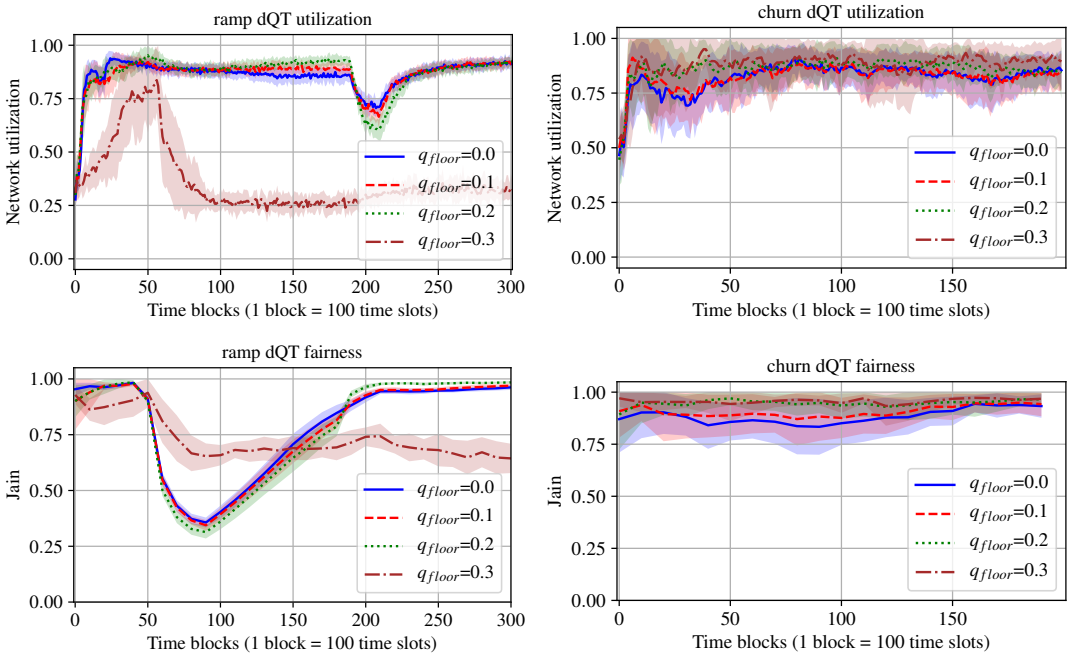


Figure 6.3: Varying the weight floor q_{floor} in the ALOHA-dQT protocol.

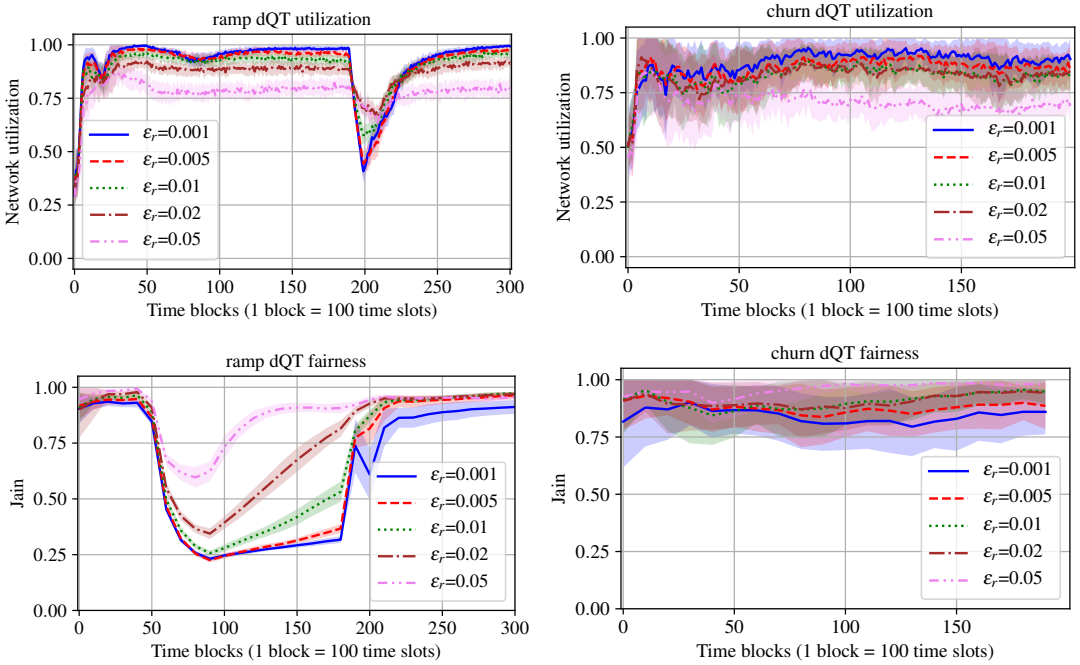


Figure 6.4: Varying the relinquishment probability ϵ_r in the ALOHA-dQT protocol.

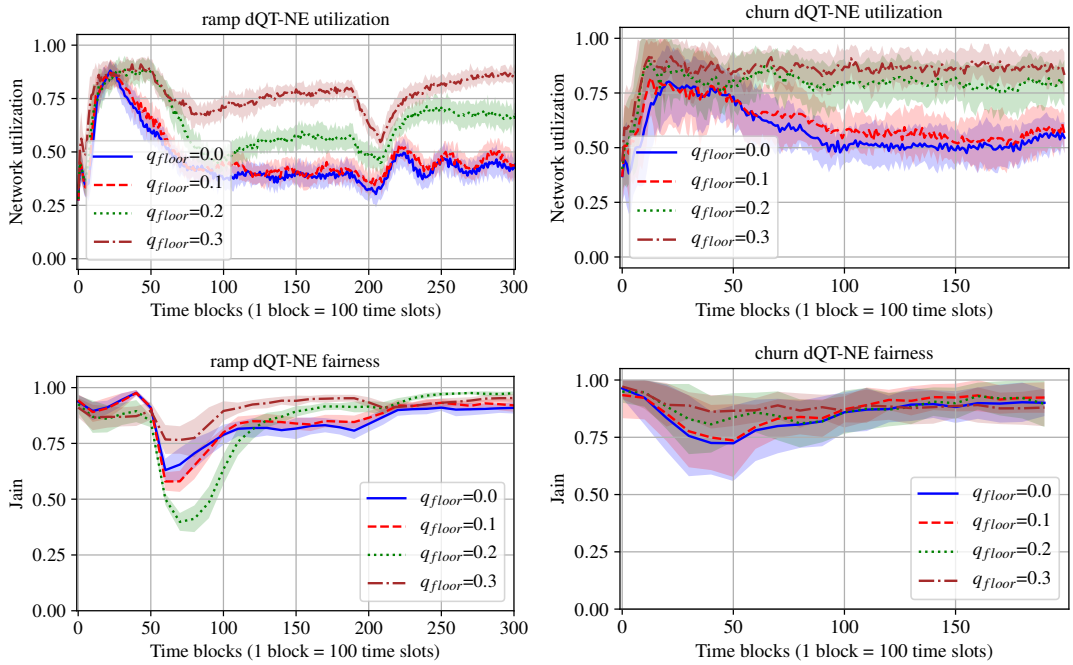


Figure 6.5: Varying the weight floor q_{floor} in the ALOHA-dQT-NE protocol.

ALOHA-dQT-NE Figures 6.5 and 6.6 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 result is that for ALOHA-dQT-NE, a higher weight floor q_{floor} is strongly beneficial, as indicated by Figure 6.5. 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 schedule weight to use them. If weights 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.

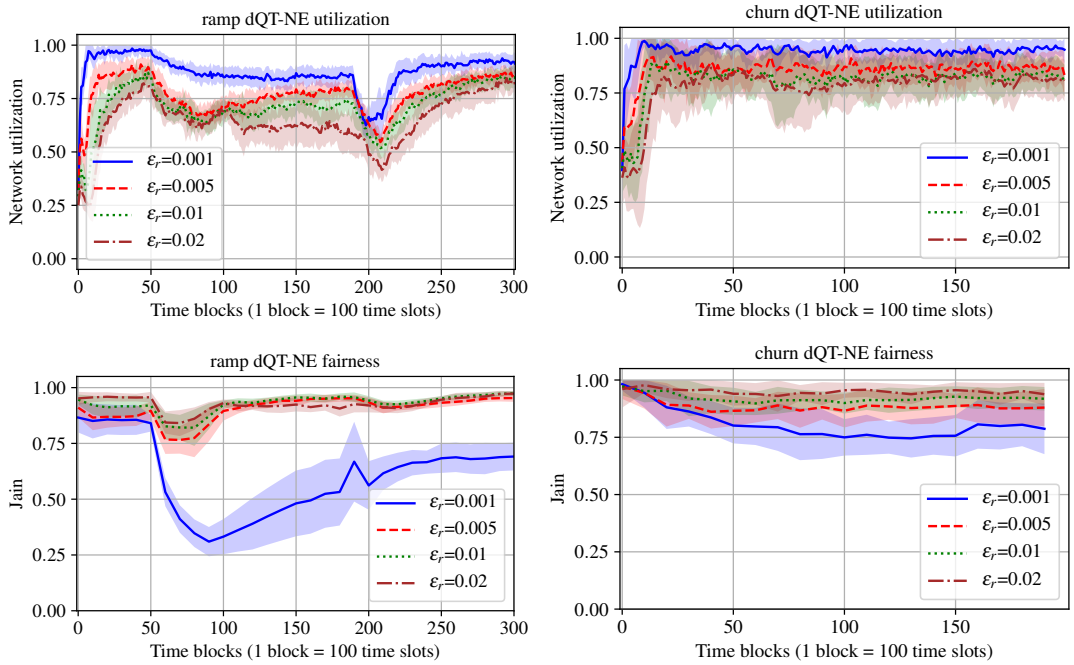


Figure 6.6: Varying the relinquishment probability ϵ_r in the ALOHA-dQT-NE protocol.

6.4 Conclusion

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 [10, 9], ALOHA-QTF [13], and the deep-RL approach of [45], 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.

Chapter 7

Final Conclusion

7.1 Summary

In this thesis, we presented four families of medium access protocols that use policy tree to learn to coordinate transmission between network nodes. They are presented in Chapter 3 to Chapter 6 respectively, in chronological order based on when they were worked on.

These four protocols all share these properties:

- They are for fully-connected single-channel time-slotted network.
- Periodic transmission schedules are arranged in a binary tree called Policy Tree. Transmission is determined by selecting one more schedules in the policy tree.
- The policy tree is updated according to the channel history. The updates help nodes select the schedules that would leave to successful transmission, and avoid selecting the schedules that would leave to collisions.

- Both network utilization and fairness of the utilization are optimized in the learning or update process.

These four protocols differ in these two major ways:

- The policy tree can be *discrete* or *quantitative*. In the former case, a schedule in the policy tree is deterministically added or removed, by using operations like demote or barge-in, this leads to AT-ALOHA (Chapter 3) and APT-ALOHA (Chapter 5). In the latter case, there is a weight associated with each schedule in the tree, and the weight determines whether a schedule is selected or not. This leads to ALOHA-QT (Chapter 4) and ALOHA-dQT (Chapter 6).

- The acknowledgement mechanism can be *immediate* or *delayed*. In immediate acknowledgement, as in the case of AT-ALOHA (Chapter 3) and ALOHA-QT (Chapter 4), the channel outcome of each time slot is immediately known the the nodes, which then results in prompt update to the policy tree. In delayed acknowledgement, as in the case of APT-ALOHA (Chapter 5) and ALOHA-dQT (Chapter 6), the channel outcome won't be known to the nodes until they have received acks from subsequent successful transmissions from peer nodes. This necessitates mechanisms to reach consensus about channel history and a delayed update to the policy tree.

All four families of policy-tree-based RL protocols outperform preceding ALOHA protocols, such as ALOHA-EB, ALOHA-Q and DRL protocol by a large margin, in terms of network utilization, fairness of the utilization distribution and the adaptation

speed. The policy tree based protocols can generally reach 80-90% network utilization in various simulation scenarios, as compared to 30-50% for ALOHA-EB and ALOHA-Q. The policy tree based protocols also outperform deep reinforcement learning based protocol [45] as the latter takes up to 10,000 time slots to converge to maximum performance whereas our protocols need only 10-100 time slots in similar setting, allowing them to quickly adapt to changing network conditions. Further more, all of the preceding RL protocols rely on immediately knowing the channel outcome after each time slot. The delayed-ack policy tree protocols presented in this thesis (Chaper 5 and 6) are the first of its kind to be usable in ad-hoc networks, where there is no central repeater to inform nodes of their transmission outcome immediately.

7.2 Comparison of all policy tree based ALOHA protocols

Figure 7.1 shows the network utilization comparison of all policy tree based protocols introduced in this thesis, in "ramp experiment" as described in Section 6.3.4. The four protocol families have performance that are comparable with each other. It's interesting to note that, during the period of new nodes joining the network ($10 \rightarrow 50$ nodes), the fairness of the protocols all dipped while the utilization remained relatively stable, whereas during the period of nodes leaving the network ($50 \rightarrow 30$ nodes), the fairness of the protocols stayed stable while the network utilization all dipped. This is because (1) when new nodes join, it takes time for them to gain network share by way of existing nodes relinquishing their transmission (thus the dip in fairness) (2) when

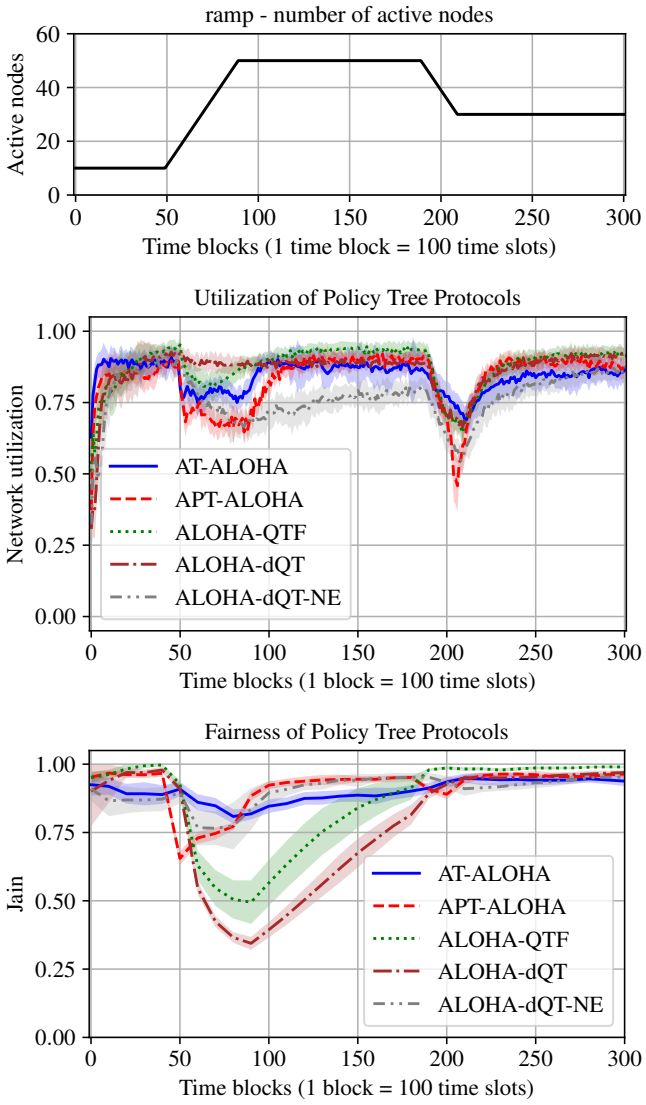


Figure 7.1: **Comparing all policy tree ALOHA protocols.** The number of active nodes go from 10 to 50 to 30. Solid line is the average of 20 runs with the colored bands showing plus/minus one standard deviation from the average.

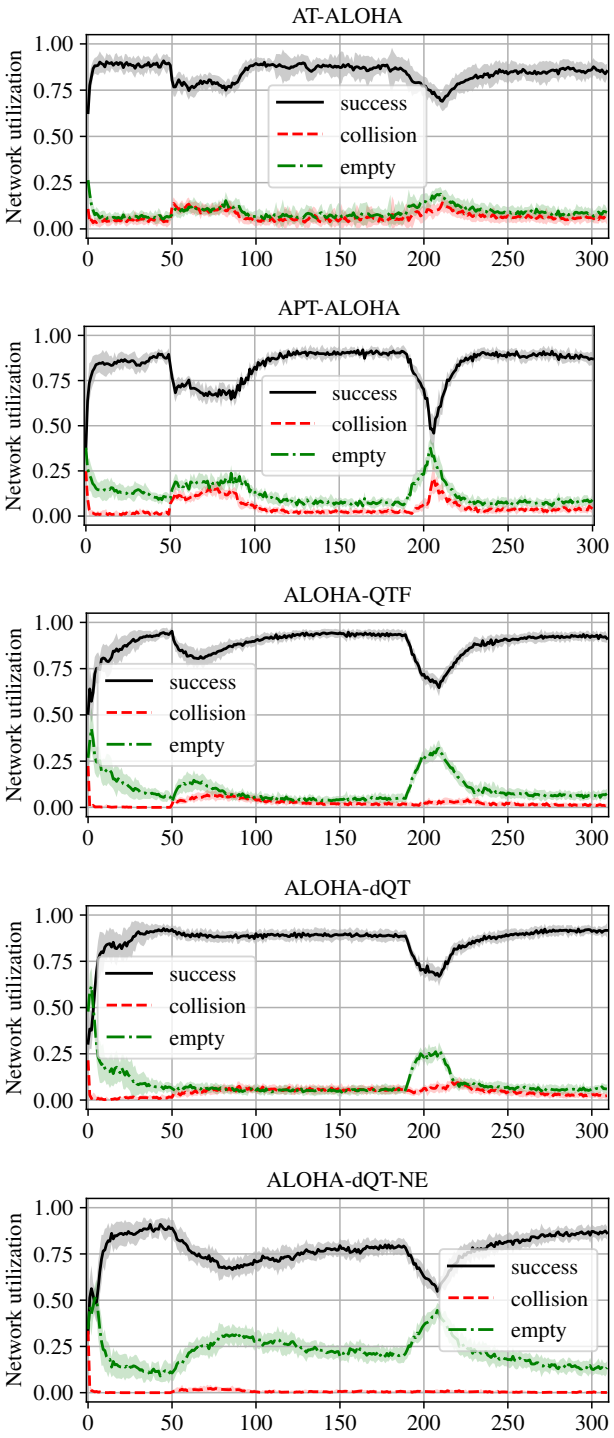


Figure 7.2: **Success, empty and collision rate of each policy tree protocol**

nodes leave, the bandwidth left by their absence are shared fairly, but that coordination process happens gradually, and thus the dip in network utilization.

Figure 7.2 shows the success, empty and collision rate of each one of the five policy protocols, in the same setting as Figure 7.1, where the number of nodes go from 10 to 50 and then back to 30. We can see that, despite the similar performance in terms of network utilization, there is a distinct difference in terms of the rate of empty and collision time slots. In discrete tree based protocols (AT-ALOHA and APT-ALOHA), the ratio of empty to collision time slot is kept constant through tuning barge-in probability p_b, and it's reflected figure. In quantitative tree baesd protocols (ALOHA-QTF, ALOHA-dQT and ALOHA-dQT-NE), empty time slot is tolerated while collisions are minimized. This difference in behavior is due to the differing policy tree update mechanism in discrete and quantitative policy trees, which will be discussed in detail in the next section.

Despite the differences in acknowledgement mechanism, the delayed acks and the delayed policy tree updates in APT-ALOHA (Chapter 5) and and ALOHA-dQT (Chapter 6) didn't impede their performance compared to the immediate-ack peers, even though the recovery from dip in utilization is slower. That is understandable given the delay in the policy tree update. All in all, policy tree based scheduling mechanism maintains coordination even with delay in it's feedback and updates.

7.3 Discrete v.s. Quantitative Tree

AT-ALOHA and ALOHA-QT (as well as their delayed ack countparts) use policy tree that is either discrete or quantitative. In the former case, schedules are selected from the tree through tree operations such as demote and barge-in, and in the latter case, schedules are selected according to their weights, which is updated through a multiplicative update (equation 4.1). The advantages, disadvantages, details and implications of the differences are discussed bellow:

Discrete policy tree could respond to network outcome faster. For example, in the case of a collision, a discrete policy tree immediately demotes the schedule responsible for the collision, which usually leads to a subsequent child schedule that transmits half the time as its parent. This immediate reduction in transmission is not necessarily true in quantitative policy tree, in which the one-time weight reduction might or might not lead to eliminating the offending schedule from future transmission. However, this disadvantage for quantitative tree can be mitigated by selecting a more negative α^- (multiplicative update factor for collisions) that would result in sufficient weight reduction. Following this point, because of the rule-based and deterministic nature of discrete policy tree, the updates are less customizable compared to quantitative tree, in which the multiplicative update factors α can be tuned according to the setting. The simplicity versus customizability can be trade offs that lead to selecting different protocols according to the demand of the situation.

Another aspect of difference between discrete vs quantitative policy tree, is

the mechanism and efficiency of learning, more specifically, how well does a policy tree integrate past events into the policy tree structure/weights which then guide coordination? In a discrete policy tree, learning is done through adopting different position in the tree, and that position information compasses all of the memory of the past transmission outcomes. In a quantitative policy tree, the memory of past events are stored in weights in each schedule, which retains a higher level of continuity from the past. In other words, in a tree that has n nodes, there are n^2 possible discrete trees, and an infinite number of possible quantitative trees. This means that, theoretically, quantitative tree is much better at storing information. One could argue that a discrete tree could also store infinite information because there is no upper limit on how large the tree can expand, however, infinite expansion in tree size is not nearly as computationally efficient as storing a continuous weight in nodes of a much smaller tree.

Here is a concrete example to illuminate the difference discussed above: consider in a quantitative policy tree, a schedule σ_x is currently not selected because its weight is not one of the highest (schedules with weights over 0.95 are selected, and this schedule in question has a weight of, say 0.8). However, after a subsequent positive update, σ_x weight can be boosted such that it will be selected next. Schedule σ_x can be considered a "candidate schedule" that could be promoted to active schedule quickly, whereas alternative schedules with weight 0.1 can't be. This differential tiers of potentially candidate schedules don't exist in discrete policy tree, as all schedules in a discrete tree are either selected, or not, there is no weight to bridge the gap in between the two states. In this way weight of the schedules stores information about the "transmission

potential" of each schedule that can be used for better coordination.

Another important difference between discrete and quantitative policy tree is the rate of empty and collision time slots. In discrete tree protocols, we implemented a mechanism to ensure that the ratio of empty to collision time slots is around 1.4, by tuning the p_b barge in probability (Section 3.1.5.), this mechanism worked as intended, and therefore in discrete tree, empty and collisions happen in tandem: whenever there is more empty time slots, p_b will change in the direction of causing more collisions, and vice versa. This mechanism ensures that empty time slots are quickly taken and collisions are quickly resolved. In quantitative tree approaches, no such mechanism exists, and it simply tries to avoid both empty and collision separately. The result of these two different mechanism can be seen in Figure 7.2: in AT-ALOHA and APT-ALOHA, empty and collision rate moves in tandem, and in ALOHA-QTF, ALOHA-dQT, and ALOHA-dQT-NE, the collision rate is always low and the rate of empty time slot mostly determines the network utilization.

The last difference between discrete versus quantitative policy tree protocols, are the ease of upgradability. If one were to make improvements on discrete tree, new tree operations can be introduced, outside the demote, barge-in, normalize and pruning operations, or that edits can be made to how each operations are done. For quantitative tree, an obvious place of improvement would be to replace the weight update equation 4.1 or 6.3 with superior ones. The former requires more extensive code changes whereas the latter only involves swapping one equation for another. This is due to that fact that quantitative tree formalized and abstracted away tree operations with mathematical

equations, whereas in discrete policy tree, the tree operations are rule-based and not unified.

7.4 Future Directions

Here are a few potential ways to improve the existing policy tree protocols, first of all, as discussed in the last section, the policy tree updating mechanism can be improved, either for discrete policy tree or the quantitative policy tree. The former can be done through innovative tree operations and the latter can be done through having a superior update equation that can better leverage the information stored in the schedule weights. A second direction of possible improvements, which haven't been explored in this thesis, is to introduce adversarial nodes. The experiments described in the performance sections of the protocols did not consider the existence of adversarial nodes that intentionally disrupt the transmission, therefore it's unknown whether they will perform well under those circumstances. Introducing adversarial nodes can push the protocols to their limits and then reveal new ways in which they can be made better. A third potential direction of improvement is through reduction and automatic learning of hyper-parameter. One caveat for the protocols presented in this thesis, is that they rely on a large number of pre-set hyperparameters, some of which need to be tuned in order to achieve optimal performance, and different protocols in different network settings need different sets of hyper parameters to perform well. This hyper parameter selection is a manual process that can be time consuming. If they can be automatically

learnt, that will also reduce the complexity and to increase robustness of the protocols.

7.5 Code

The github repository: https://github.com/MollyZhang/PolicyTreeProtocol contains the python code to reproduce the results for AT-ALOHA and ALOHA-QT. The code for delayed ack versions APT-ALOHA and ALOHA-dQT are not included yet due to pending patent application.

Bibliography

[1] Norman Abramson. The throughput of packet broadcasting channels. *IEEE Transactions on Communications*, 25(1):117–128, 1977.

[2] T. Adame, A. Bel, B. Bellalta, J. Barcelo, and M. Oliver. Ieee 802.11ah: the wifi approach for m2m communications. *IEEE Wireless Communications*, 21(6):144–152, 2014.

[3] Peter Alvaro, Neil Conway, Joseph M. Hellerstein, and William R. Marczak. Consistency Analysis in Bloom: a CALM and Collected Approach. In *CIDR*, pages 249–260, 2011.

[4] Lichun Bao and JJ Garcia-Luna-Aceves. A new approach to channel access scheduling for ad hoc networks. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 210–221, 2001.

[5] Lichun Bao and JJ Garcia-Luna-Aceves. Hybrid channel access scheduling in ad hoc networks. In *10th IEEE International Conference on Network Protocols, 2002. Proceedings.*, pages 46–57. IEEE, 2002.

[6] Olivier Bousquet and Manfred K. Warmuth. Tracking a small set of experts by mixing past posteriors. *Journal of Machine Learning Research*, 3(Nov):363–396, 2002.

[7] John Capetanakis. Tree algorithms for packet broadcast channels. *IEEE transactions on information theory*, 25(5):505–515, 1979.

[8] Ursula Challita, Li Dong, and Walid Saad. Deep learning for proactive resource allocation in lte-u networks. In *European wireless technology conference*, 2017.

[9] Yi Chu, Selahattin Kosunalp, Paul D. Mitchell, David Grace, and Tim Clarke. Application of reinforcement learning to medium access control for wireless sensor networks. *Engineering Applications of Artificial Intelligence*, 46:23–32, 2015.

[10] Yi Chu, Paul D. Mitchell, and David 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.

[11] Neil Conway, William R. Marczak, Peter Alvaro, Joseph M. Hellerstein, and David Maier. Logic and lattices for distributed programming. In *Proceedings of the Third ACM Symposium on Cloud Computing*, pages 1–14, 2012.

[12] Luca de Alfaro, Molly Zhang, and JJ Garcia-Luna-Aceves. Approaching fair collision-free channel access with slotted aloha using collaborative policy-based reinforcement learning. In *2020 IFIP Networking Conference (Networking)*, pages 262–270. IEEE, 2020.

[13] Luca de Alfaro, Molly Zhang, and JJ Garcia-Luna-Aceves. Approaching fair collision-free channel access with slotted aloha using collaborative policy-based reinforcement learning. In *IEEE IFIP Networking Conference*, 2020.

[14] JJ Garcia-Luna-Aceves and Ashok N Masilamani. Nomad: Deterministic collision-free channel access with channel reuse in wireless networks. In *2011 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 1–9. IEEE, 2011.

[15] JJ Garcia-Luna-Aceves and Ashok N Masilamani. Using radio connectivity to define transmission schedules in multihop wireless networks. In *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*, pages 434–442. IEEE, 2014.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] David P. Helmbold, Darrell DE Long, and Bruce 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.

[18] Mark Herbster and Manfred K. Warmuth. Tracking the best expert. *Machine learning*, 32(2):151–178, 1998.

[19] Huaizhou Shi, R. Venkatesha Prasad, Ertan Onur, and I. G. M. M. Niemegeers. Fairness in Wireless Networks:Issues, Measures and Challenges. *IEEE Communications Surveys & Tutorials*, 16(1):5–24, 2014.

[20] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 1984.

[21] Gentian Jakllari, Mike Neufeld, and Ram 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. IEEE, 2012.

[22] Jeong DG and Jeon WS. Performance of an exponential backoff scheme for slotted-aloha protocol in local wireless environment. *IEEE Transactions on Vehicular Technology*, 44(3):470–479, Aug 1995.

[23] Ehsan Ebrahimi Khaleghi, Cedric Adjih, Amira Alloum, and Paul 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.

[24] Leonard Kleinrock. *Queueing systems. Volume I: theory.* wiley New York, 1975.

[25] Gianluigi Liva. Graph-based analysis and optimization of contention resolution diversity slotted aloha. *IEEE Transactions on Communications*, 59(2):477–487, 2010.

[26] Ashok N Masilamani and JJ Garcia-Luna-Aceves. Scheduled channel access using geographical classification. In *2013 International Conference on Computing, Networking and Communications (ICNC)*, pages 790–796. IEEE, 2013.

[27] Kais Mekki, Eddy Bajic, Frederic Chaxel, and Fernand Meyer. A comparative study of lpwan technologies for large-scale iot deployment. *ICT Express*, 5(1):1 – 7, 2019.

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

[29] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[30] Oshri Naparstek and Kobi Cohen. Deep multi-user reinforcement learning for dynamic spectrum access in multichannel wireless networks. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–7. IEEE, 2017.

[31] Hiromi Okada, Y Igarashi, and Y Nakanishi. Analysis and application of framed aloha channel in satellite packet switching networks-fadra method. *Electronics Communications of Japan*, 60:72–80, 1977.

[32] Enrico Paolini, Gianluigi Liva, and Marco Chiani. Coded slotted aloha: A graph-based method for uncoordinated multiple access. *IEEE Transactions on Information Theory*, 61(12):6815–6832, 2015.

[33] Subramanian Ramanathan and Errol L Lloyd. Scheduling algorithms for multihop radio networks. *IEEE/ACM Transactions on networking*, 1(2):166–177, 1993.

[34] George F. Riley and Thomas R. Henderson. *The ns-3 Network Simulator*, pages 15–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[35] Lawrence G. Roberts. ALOHA packet system with and without slots and capture. *ACM SIGCOMM Computer Communication Review*, 5(2):28–42, 1975.

[36] Frits Schoute. Dynamic frame length aloha. *IEEE Transactions on communications*, 31(4):565–568, 1983.

[37] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[38] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[39] Andrew S. Tanenbaum and David Wetherall. *Computer networks*. Prentice hall, 1996.

[40] Zhenyu Tang and JJ Garcia-Luna-Aceves. A protocol for topology-dependent transmission scheduling in wireless networks. In *WCNC. 1999 IEEE Wireless Com-*

munications and Networking Conference (Cat. No. 99TH8466), volume 3, pages 1333–1337. IEEE, 1999.

[41] Dimitrios J Vergados, Natalia Amelina, Yuming Jiang, Katina Kralevska, and Oleg Granichin. Local voting: Optimal distributed node scheduling algorithm for multihop wireless networks. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1014–1015. IEEE, 2017.

[42] Shangxing Wang, Hanpeng Liu, Pedro Henrique Gomes, and Bhaskar Krishnamachari. Deep reinforcement learning for dynamic multichannel access in wireless networks. *IEEE Transactions on Cognitive Communications and Networking*, 4(2):257–265, 2018.

[43] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[44] Zhenyu Yang and JJ Garcia-Luna-Aceves. Hop-reservation multiple access (hrma) for ad-hoc networks. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, volume 1, pages 194–201. IEEE, 1999.

[45] Yiding Yu, Taotao Wang, and Soung Chang Liew. Deep-reinforcement learning multiple access for heterogeneous wireless networks. *IEEE Journal on Selected Areas in Communications*, 2019.

[46] Molly Zhang, Luca de Alfaro, and JJ Garcia-Luna-Aceves. An adaptive tree algorithm to approach collision-free transmission in slotted aloha. In *Proceedings of the Workshop on Network Meets AI & ML*, pages 56–61, 2020.

[47] Molly Zhang, Luca de Alfaro, and JJ Garcia-Luna-Aceves. Using reinforcement learning in slotted aloha for ad-hoc networks. In *Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 245–252, 2020.

[48] Molly Zhang, Luca de Alfaro, Marc Mosko, Colin Funai, Tim Upthegrove, Bishal Thapa, Daniel Javorsek, and JJ 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)*, pages 138–146. IEEE, 2020.