Adaptive Policy Tree Algorithm to Approach Collision-Free Transmissions in Slotted ALOHA

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

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

Marc Mosko
PARC
mmosko@parc.com

Colin Funai
Raytheon BBN Technologies
colin.funai@raytheon.com

Tim Upthegrove
Raytheon BBN Technologies
tim.upthegrove@raytheon.com

Bishal Thapa
Raytheon BBN Technologies
bishal.thapa@raytheon.com

Daniel Javorsek
US Air Force
daniel.javorsek@us.af.mil

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

***Abstract*—A new adaptive transmission protocol is introduced to improve the performance of slotted ALOHA. Nodes use known periodic schedules as base policies with which they collaboratively learn how to transmit periodically in different time slots so that packet collisions are minimized. The Adaptive Policy Tree (APT) algorithm is introduced for this purpose, which results in APT-ALOHA. APT-ALOHA does not require the presence of a central repeater and uses explicit acknowledgements to confirm the reception of packets. It is shown that nodes using APT-ALOHA quickly converge to transmission schedules that are virtually collision-free, and that the throughput of APT-ALOHA resembles that of TDMA, where slots are pre-allocated to nodes. In particular, APT-ALOHA attains a successful utilization of time slots - over 70% on saturation mode.**

I. Introduction

The ALOHA channel [1] was the first example of contention-based medium access control (MAC) protocols. Its key feature is simplicity; a node with a packet to send simply transmits. This inherent simplicity makes ALOHA and its variants an attractive choice for channel access in such untethered networks as underwater acoustic networks, satellite networks, space networks, and wireless networks in which hidden-terminal interference is prevalent. However, the simplicity of ALOHA comes at the price of performance, with a maximum throughput of only 18% of the bandwidth available for transmission to active receivers.

As Section II summarizes, many approaches have been proposed over the years to improve the performance of ALOHA, and all of them have been based on the assumption that nodes transmit at the beginning of time slots in order to reduce multiple-access interference. Even when transmissions are organized into time-slots, the fraction of time slots that can be successfully used is limited to twice the pure ALOHA limit, or about 37%, unless the nodes somehow coordinate their transmissions.

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

Section II describes prior work on the use of reinforcement learning aimed at allowing nodes to learn how to coordinate their use of time slots. These protocols attain remarkable channel efficiency and approach the performance of TDMA with fixed-schedules. However, a major limitation of all these reinforcement-learning methods, including recent proposals based on deep learning [2], is that they require *immediate acknowledgements* to operate, such that a transmitting node knows whether or not its transmission is successful at the end of the time-slot in which the transmission takes place. This assumption is viable in a centralized setting like the one assumed for the original work on ALOHA and slotted ALOHA, where every transmission is immediately rebroadcast or acknowledged by a central node, and an up-link and down-link channel are used. However, this is not realistic in ad-hoc scenarios, where a single shared channel is used and half-duplex nodes must switch their radios from receive to transmit mode when transmitting packets, and switch the radios back to receive mode once their transmissions are over. Thus, nodes cannot detect collisions or corruption of their own transmission.

In this paper, we extend the reinforcement-learning approach proposed in [3] to make it suitable to general time-slotted ad-hoc networks. Sections III and IV describe the resulting algorithm, which we call APT-ALOHA for *Adaptive Policy Tree ALOHA*. The approach is based on associating with each node a tree of periodic schedules of different periods. The nodes adapt to each other's transmissions by selecting schedules that minimize conflicts. The tree structure guides adaptation, and enables high network utilization under a wide range of network conditions.

Adapting reinforcement learning protocols to delayed acknowledgements requires the introduction of a new acknowledgement scheme, as well as new methods to perform the learning and to ensure the fairness of bandwidth use. To this end, we introduce an acknowledgement scheme based on a gossip protocol, and we propose new techniques for adapting the transmission policies and achieving fair bandwidth sharing in spite of the delayed acknowledgement information.

Section V analyzes the performance of APT-ALOHA using detailed ns-3 simulations that take into account the nodes' spacial distribution, interference, propagation delays, signal to noise levels, and capture (different receivers receiving different packets). The results of the simulation experiments show that APT-ALOHA achieves channel utilization exceeding 70%, successfully adapting to variable network conditions. This high utilization is achieved while dividing the available bandwidth fairly among the active network nodes. This is far better throughput and fairness than ALOHA-EB, and it closely matches the throughput achieved via AT-ALOHA and ALOHA-QT, in spite of the reliance of the two latter protocols on immediate acknowledgements.

Section VI presents our conclusions.

II. Related Work

Roberts [4] introduced slotted ALOHA to improve on the performance of pure ALOHA by forcing transmissions to occur at the beginning of time slots. The key benefit derived from slotted ALOHA, over pure ALOHA, is that it reduces the time during which transmissions are vulnerable to multiple-access interference (MAI) by half.

Jeong and Jeon [5] 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.

If every node transmits in a time-slot with the same probability p, the optimal value for p is $1/n$, where n is the number of active nodes, and as n grows, the network utilization is bounded by $1/e \approx 0.37$. To break through this bandwidth bound, it is necessary to achieve a deeper level of coordination, in which nodes adapt to each other's behavior so that most transmission slots can be utilized with only a few collisions.

One way to achieve this coordination is to endow each node with transmission policies consisting of the union of periodic schedules that reduce conflict, along with a way to choose among such policies. We consider here periodic schedules of the form (i, m), where schedule (i, m) prescribes transmitting every i-th slot in a period of length 2^m. These schedules can be arranged in a *policy tree*, where the schedule (i, m) has as descendants the schedules $(i, m + 1)$, and $(i + 2^m, m + 1)$. Sibling schedules do not conflict, and descendant schedules transmit in a subset of the slots of their ancestors.

In ALOHA-QT [6], the nodes choose among these schedules using a reinforcement learning algorithm that associates with each schedule a "quality" that represents how successful the schedule has been in prescribing conflict-free transmissions. ALOHA-QT was shown to lead to high network utilization, typically above 80% in steady-state, with few empty slots and even fewer collisions.

An alternative scheme, termed AT-ALOHA [3] uses an adaptive algorithm for the selection of schedules that deterministically considers, at each step, a subset of *active* schedules which are followed The set of active schedules is updated according to the outcome of each transmission slot.

The schedule tree used in AT-ALOHA, ALOHA-QT, and in the present paper are closely related to the collision resolution scheme introduced by Capetanakis et al. [7]. While the schedule tree in [7] is used to resolve every individual collision as it occurs, the algorithms developed in these later papers use the trees to learn a long-term transmission policy that minimizes collisions. The schedule tree has been used also by Jakllari et al. [8], who propose a bandwidth reservation scheme over time-slotted channels.

Another approach based on reinforcement learning is the ALOHA-Q protocol (framed slotted ALOHA with Q-learning) by Chu et al. [9], [10]. In ALOHA-Q, the time slots are grouped in frames of fixed length M. Each node has available M schedules, where schedule $i \in [1, \ldots, M]$ prescribes transmitting in the i-th slot of the frame, and keeps track of the *quality* q_i of each such schedule. When a new frame starts, the node selects the schedule with the highest quality, and updates the schedule quality according to the success or failure (due to collisions) of the schedule. In case of collisions, the nodes resort to skipping frames with a backoff mechanism. Similarly to AT-ALOHA and ALOHA-QT, the protocol is presented for networks were transmission outcomes are available immediately due to repeaters or similar mechanisms.

Deep reinforcement learning (DRL) has been recently proposed by Yu el al. [2] to overcome the limitations of ALOHA-Q stem from relying on a small, fixed set of policies. In DRL, a neural network is trained to predict the value (in our case, probability of success) or each action (in our case, transmit or wait) as a function of the system history (in our case, the time-slot contents in the last n time-slots, for some $n > 0$). Due to the large state-spaces needed to represent network histories, neural networks take a long time to learn effective strategies. Adaptation has been demonstrated only for networks up to two DRL nodes, and even for those, adaptation required tens of thousands of time-slots (alongside a complex node architecture).

What is critical to point out is that all of this prior work relies on the assumption of immediate acknowledgements.

III. The Schedule Tree

APT-ALOHA is a protocol designed for nodes that share a time slotted transmission channel. To share the channel efficiently, the nodes coordinate their transmissions by transmitting according to one or more periodic schedules with periods that are powers of 2. We assume that every node has a clock t that counts the number of time slots. A periodic 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). 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 with root $(0,0)$, where the schedule (i,m) has $(i,m+1)$ and $(i+2^m, m+1)$ as children. A child schedule transmits in only half the time slots as its parent. 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 $T(\pi) = \bigcup_{s \in \pi} T(s)$. We require policies to be in normal form: a policy should not contain schedules that are one the descendant of the other, nor sibling schedules that can be merged into a single schedule. Figure 1 depicts the schedule tree, along with a policy in normal form.

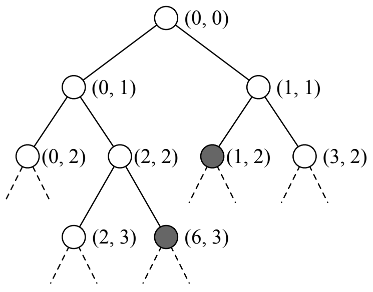


Fig. 1: A (partial) depiction of the tree of all schedules. The dark nodes in the tree correspond to policy $\pi = \{(1,2),(6,3)\}$. That policy achieves a throughput of $1/2^2 + 1/2^3 = 1/4 + 1/8 = 0.375$ in absence of collisions.

A node updates a policy 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.

Nodes start with a simple policy consisting of a single active schedule, and evolve their policy in response to the empty slots, collisions, and packet acknowledgements they receive using the above operations. Since the set of active schedules evolve in time, and as the schedules have different periods, the APT-ALOHA protocol is *frameless*.

We present in this section the operations, and in the next their use by the protocol. The four operations were introduced in [3]; we have refined here the definitions of demotion and barge-in to allow for a faster adaptation, necessary under the delayed feedback provided by delayed acknowledgements.

Demotion: The procedure *demote*(π, t, k) is illustrated in Figure 2. If $t \notin T(\pi)$, then *demote*(π, 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

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

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 IV-C.

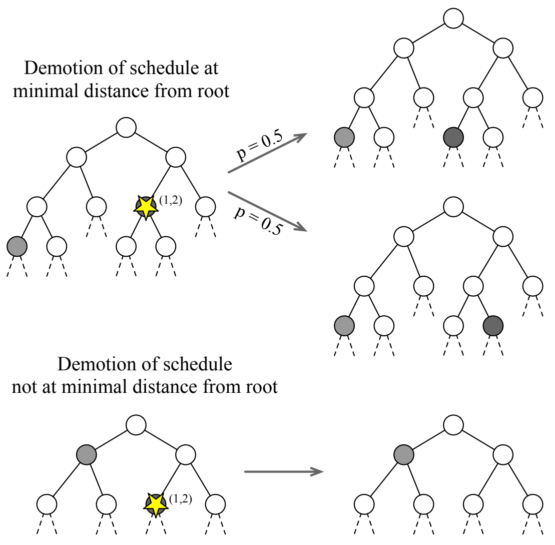


Fig. 2: Two cases of of demoting the starred schedule $(1,2)$ in the policy via *demote*$(\pi, 1, 3)$.

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*(π, 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.

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 IV-C1.

Policy normalization: The normalization operation *normalize*(π) performs descendant elimination and sibling merging:

- *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 $(i, m-1)$.

Descendant elimination and sibling merging do not modify the transmission times of the policy.

Policy pruning: After the policy is normalized, the policy is *pruned* to enforce a maximum number of active schedules. The pruning is performed in two steps, first limiting the policy depth in the tree, then the number of active schedules in it, as illustrated in Figure 3. The procedure $prune(\pi, \Delta, M)$ is defined as follows.

- Let $k = \min\{m \mid (i, m) \in \pi\}$ be the top level of an active schedule. First, we prune all schedules of level below $k + \Delta$, letting $\pi := \{(i, m) \mid (i, m) \in \pi \wedge m \leq k + \Delta\}$.
- Second, we prune π to ensure it contains at most M schedules. If $|\pi| \leq M$, we leave pi unchanged. Otherwise, let $n_k = |\{(i, m) \in \pi \mid m \leq k\}|$, and let k be the largest integer such that $n_k \leq m$. We remove from π all schedules (i, m) with $m > k + 1$, and we randomly remove $M - n_k$ of the schedules at level $k + 1$, that is, of the form $(j, k + 1)$ for some j.

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 $\pi :=$ *prune*(*normalize*$(\pi), 2, 10)$.

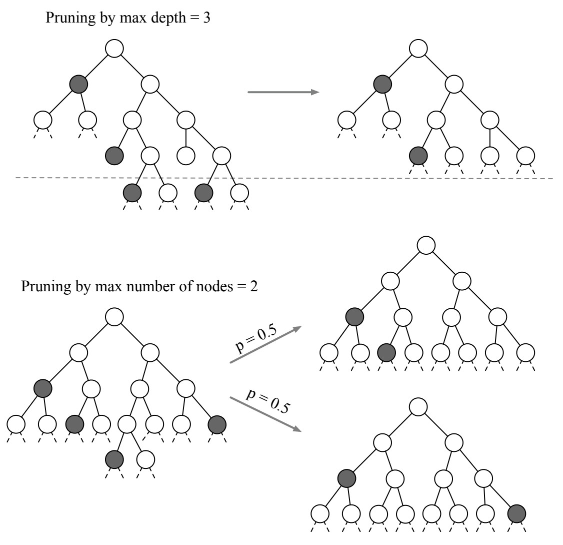


Fig. 3: Policy pruning operations: limiting the depth (top), and limiting the number of nodes (bottom).

IV. The APT-ALOHA Protocol

The APT algorithm is presented schematically in Algorithm 1; we describe below its structure and behavior.

A. 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 $\hat{N}$

```
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;
    $\pi$: APT policy, with schedule labeling $\ell$;
    $A_{pending}, A_{out}$: lists of pending and outgoing
      acknowledgements;
    $\eta$: expiry time for acknowledgements;
    $\hat{N}$: estimated number of active network nodes (see
      Sec IV-C1);
    $p_k$: kindness probability (see Sec IV-C2);
Channel Variables:
    $h \in \{S, E, C, R\}$: channel state at the end of a time slot.
Initialization:
    $t := 0$; $p_b := 0.1$; $\pi := choice\{(0, 1), (1, 1)\}$;
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 $1/\hat{N}$:
          $\pi = bargein(\pi, t, \lfloor \log_2(\hat{N} - 1) \rfloor)$
    else
        $p_k := \max(q_k, p_k/\alpha)$
    foreach $t' \in A_{pending}$ do (ack expiration)
        if $t - t' > \eta(t')$ then
            $ldemote(\pi, t', \lceil \log_2 \hat{N} \rceil)$;
            $A_{pending} := A_{pending} \setminus \{t'\}$
    $\pi := prune(normalize(\pi), 2, 10)$;
    $t := t + 1$;
    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
                  $ldemote(\pi, t', \lceil \log_2 \hat{N} \rceil)$
            else (demotion due to collision)
                $ldemote(\pi, t', \lceil \log_2 \hat{N} \rceil)$
        if $s \neq s'$ then (ack for another node)
            if $t' \in A_{pending}$ then (virtual collision sender)
                $A_{pending} := A_{pending} \setminus \{t'\}$;
                $ldemote(\pi, t', \lceil \log_2 \hat{N} \rceil)$
            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 1: The APT Algorithm.

of the number of other active network nodes, and a *kindness* probability p_k, tuned as described in Sec IV-C.

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 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 policy (i, m) that caused transmission might have been replaced by another policy (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 policies 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 (by a demotion, barge-in, or renormalization) 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*(π, 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.

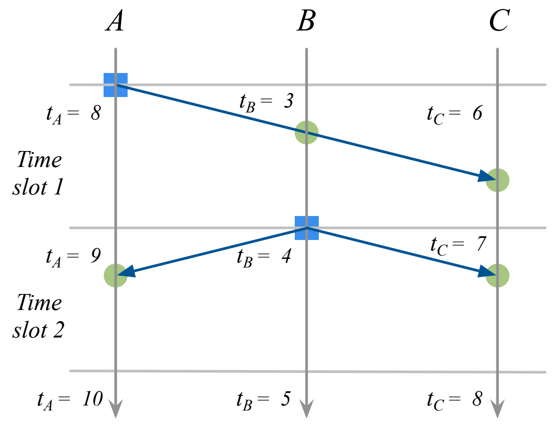
Time slot status: At the end of each time slot, the node's radio communicates to the protocol the status $h \in \{D, E, C, R\}$ of the time slot, where:

- 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 policies, 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	$\emptyset$	$\{(B, 7, \text{T})\}$	

Fig. 4: 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 4. 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 $(A, 3, \mathrm{T})$ in B's outgoing queue is transmitted in relative time as $(A, -1, \mathrm{T})$. 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 $(A, 6, \mathrm{T})$ when it hears $(A, -1, \mathrm{T})$ from B: as acknowledgements are a gossip protocol, C no longer needs to acknowledge A's packet in time slot 1.

B. 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.

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 IV-C1.
- $h = C$ (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 IV-C1.

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.

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 $b = \mathrm{T}$ and there is $(s'', t', \mathrm{T}) \in A_{out}$ 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 $(s'', t', \mathrm{T}) \in A_{out}$ with (s'', t', F).
- *Removal of redundant acknowledgements.* If $b = \mathrm{T}$, the acknowledgement (s', t', T), if found in $A_{pending}$, is removed, as the packet has already been acknowledged. If $b = \mathrm{F}$, 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 $b = \mathrm{T}$, 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 collisions, time slots are not forever allocated to the same node, helping to make the protocol fair in the long term.
  - If $b = \mathrm{F}$, the node accounts for the collision by executing *ldemote*$(\pi, t', \lceil \log_2 \hat{N} \rceil)$.

C. 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.

1) Network 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 teh 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.

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.

V. Performance Evaluation

We simulate the performance of APT-ALOHA in the NS3 simulator [11]. 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) [12], [13]. 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 [5], and with AT-ALOHA [3], ALOHA-QT [6], and ALOHA-Q [9], [10] 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.

A. 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. To compute the metrics, we group time slots into blocks of 100, and we compute the fraction of Empty, Success, and Collision slots in a block as our main performance benchmarks.

We also compute Jain's fairness index [14, p. 36] for the successful transmissions. For n active nodes, let x_i be the number of successful transmissions by node i in a given interval, $1 \leq i \leq n$. The Jain's fairness index is computed as $\left(\sum_{i=1}^{n} x_i\right)^2 / \left(n \sum_{i=1}^{n} x_i^2\right)$, and has value between $1/n$ and 1; the value is 1 for a perfectly fair distribution $x_1 = \cdots = x_n$, and $1/n$ when only one of the x_i is non-zero. 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.

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.

B. APT-ALOHA Compared with ALOHA-EB

Fig. 5 shows the simulation scenario, which is made up of 5 segments, as shown in Fig. 5(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 I 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).

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

The bulk of the channel time (61.3% - 83.0%) is spent in the collision state, even during the steady-state periods.

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. 5c shows the Jain's Fairness index for the two protocols.

C. 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. APT-ALOHA

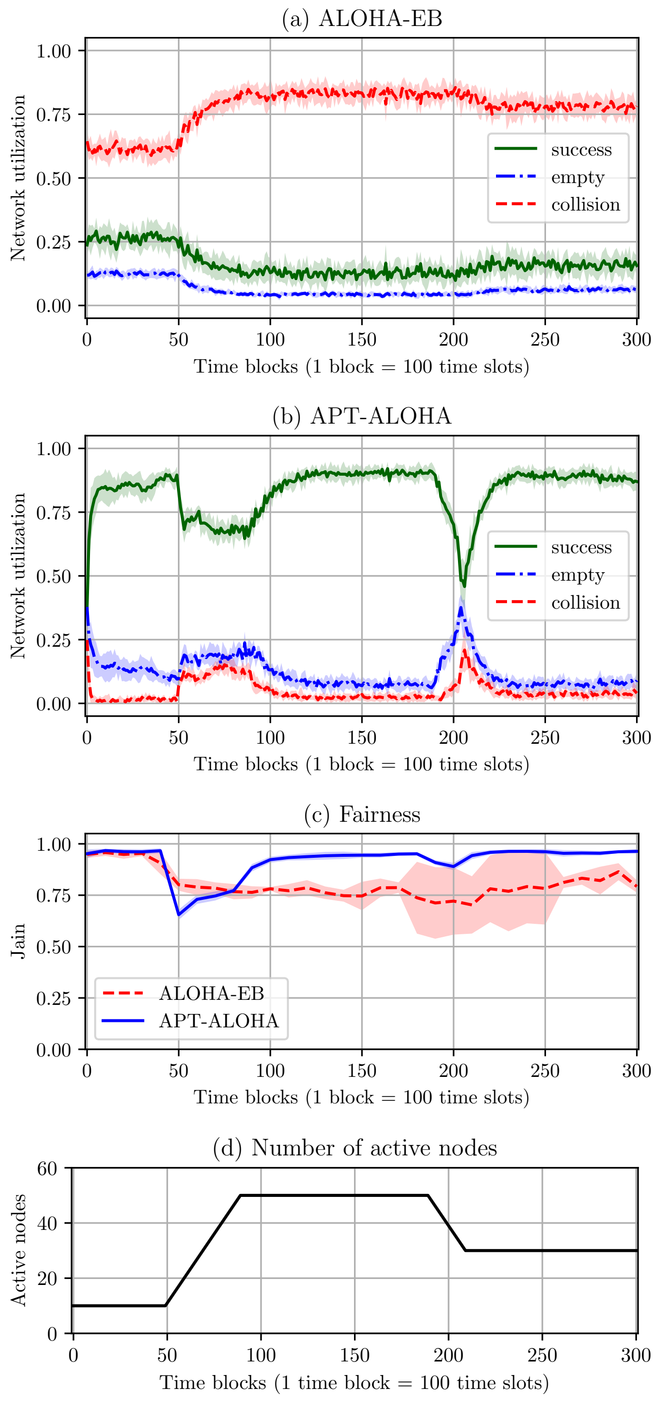


Fig. 5: Network utilization in the ramp experiment for ALOHA-EB (a) and APT-ALOHA (b), and Jain's Fairness (c). The number of active nodes is shown in (d).

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 6. 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 number 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.

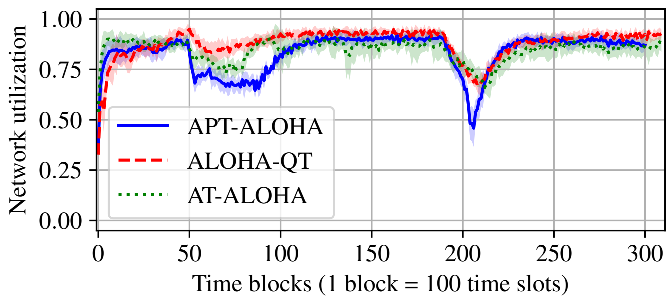


Fig. 6: Network utilization of ramp experiment for APT-ALOHA, ALOHA-QT and AT-ALOHA performs similarly

D. 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.

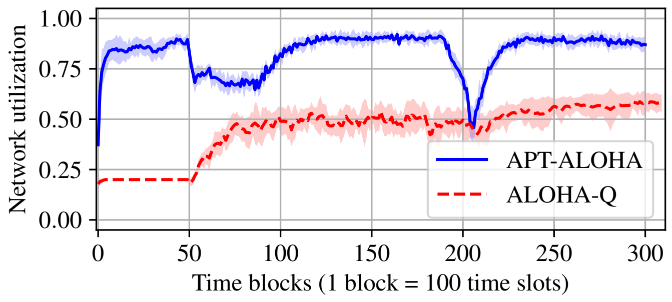


Fig. 7: Network utilization comparison of APT-ALOHA and ALOHA-Q in ramp experiment

The comparison results are provided in Figure 7. 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 [15, Section 5.4.1], an adaptation period of the order of hundreds of thousands of time slots would be needed to reach such utilization.

VI. Conclusions

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.

References

[1] N. Abramson, "The throughput of packet broadcasting channels," *IEEE Transactions on Communications*, vol. 25, no. 1, pp. 117–128, 1977.
[2] Y. Yu, T. Wang, and S. C. Liew, "Deep-reinforcement learning multiple access for heterogeneous wireless networks," *IEEE Journal on Selected Areas in Communications*, 2019.
[3] M. Zhang, L. de Alfaro, and J. Garcia-Luna-Aceves, "Collision-free channel access with delayed acknowledgements using collaborative policy-based reinforcement learning," in *ACM SIGCOMM Conference, NetAI Workshop*, 2020.
[4] L. G. Roberts, "ALOHA packet system with and without slots and capture," *ACM SIGCOMM Computer Communication Review*, vol. 5, no. 2, pp. 28–42, 1975.
[5] Jeong DG and Jeon WS, "Performance of an exponential backoff scheme for slotted-aloha protocol in local wireless environment," *IEEE Transactions on Vehicular Technology*, vol. 44, no. 3, pp. 470–479, Aug 1995.
[6] L. de Alfaro, M. Zhang, and J. Garcia-Luna-Aceves, "Approaching fair collision-free channel access with slotted aloha using collaborative policy-based reinforcement learning," in *IEEE IFIP Networking Conference*, 2020.
[7] J. Capetanakis, "Tree algorithms for packet broadcast channels," *IEEE transactions on information theory*, vol. 25, no. 5, pp. 505–515, 1979.
[8] G. Jakllari, M. Neufeld, and R. Ramanathan, "A framework for frameless tdma using slot chains," in *2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012)*. IEEE, 2012, pp. 56–64.
[9] Y. Chu, P. D. Mitchell, and D. Grace, "ALOHA and q-learning based medium access control for wireless sensor networks," in *2012 International Symposium on Wireless Communication Systems (ISWCS)*. IEEE, 2012, pp. 511–515.
[10] Y. Chu, S. Kosunalp, P. D. Mitchell, D. Grace, and T. Clarke, "Application of reinforcement learning to medium access control for wireless sensor networks," *Engineering Applications of Artificial Intelligence*, vol. 46, pp. 23–32, 2015.
[11] G. F. Riley and T. R. Henderson, *The ns-3 Network Simulator*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34. [Online]. Available: https://doi.org/10.1007/978-3-642-12331-3_2
[12] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "A comparative study of lpwan technologies for large-scale iot deployment," *ICT Express*, vol. 5, no. 1, pp. 1 – 7, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405959517302953
[13] T. Adame, A. Bel, B. Bellalta, J. Barcelo, and M. Oliver, "Ieee 802.11ah: the wifi approach for m2m communications," *IEEE Wireless Communications*, vol. 21, no. 6, pp. 144–152, 2014.
[14] R. Jain, *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, 1991.
[15] Y. Chu, "Application of reinforcement learning on medium access control for wireless sensor networks," Ph.D. dissertation, University of York, 2015.