A Reinforcement Learning-based Data-Link Protocol for Underwater Acoustic Communications

Valerio Di Valerio, Chiara Petrioli
University of Rome "La Sapienza"
Dept. of Computer Science
{divalerio, petrioli}@di.uniroma1.it

Loreto Pescosolido
University of Rome "La Sapienza"
DIET Dept.
loreto.pescosolido@uniroma1.it

Mihaela Van Der Shaar
University of California Los Angeles
Dept. of Electrical Engineering
mihaela@ee.ucla.edu

ABSTRACT

We consider an underwater acoustic link where a sender transmits a flow of packets to a receiver through a channel with time varying quality. We address the problem of scheduling packets transmission, forward error correction (FEC) code selection, and channel probing to achieve the best trade-off between energy consumption and latency. Unlike previous works, which assume complete knowledge of the statistics of the underwater acoustic environment, we make the protocol learn the optimal behavior based on experience, without relying on any prior knowledge on the environment. We design a Reinforcement-Learning (RL)-based protocol which learns how to minimize a cost function which is a combination of delay and energy consumption, at the same time ensuring packet delivery. Starting from a basic Q-learning strategy, we design two learning algorithms to speed up learning time, and compare the performance of the proposed solutions with the Q-learning-based strategy and with an aggressive strategy which always transmits all the packets in the buffer. The results show that the proposed techniques outperform the aggressive policy and Q-learning, and are successful in achieving good tradeoffs between energy consumption and packet delivery latency (PDL).

Keywords

Underwater communications, underwater networks, adaptive protocols, reinforcement learning

1. INTRODUCTION

Underwater acoustic channels are characterized by high time variability [1] which demand for adaptive data-link layer protocols, using more or less aggressive transmission schemes and different FEC codes depending on the channel conditions. Most of the research efforts devoted so far to the design of adaptive data link schemes for underwater sensor networks (UWSNs) have focused on adaptive or rateless error correction codes [2, 3], or hybrid incremental redundancy ARQ protocols [4]. In this work, we investigate an alternative path consisting in the use of Reinforcement Learning (RL) [5] to make the transmitter *learn* about the channel behavior and adapt its transmission strategy accordingly. While conventional approaches basically "react" to the channel behavior, e.g. by the incremental addition of redundancy when packets are not correctly received, our strategy does *proactive* adaptation by forecasting future changes.

Protocols based on RL have been proposed for wireless communications [6]. For underwater communications, routing layer RL-based protocols have been proposed, e.g., in [7,8]. However, to the best of our knowledge, there have not been works proposing the use of RL techniques for UWSNs at the data-link layer.

RL is an attractive technique for optimizing the data-link layer of underwater acoustic channels as some of the most significant channel variations take place with a sufficiently low frequency to let the transmitter learn and exploit effective scheduling policies tailored to each situation.Our proposed protocol provides the transmitter with an optimal policy on whether buffered packets should be transmitted immediately or delayed, what should be the burst size, and what the best FEC code to use for each transmission. The overall objective is to deliver generated packets while minimizing a cost function which combines energy consumption and latency.

In an environment which can be considered stationary for a limited amount of time (e.g., few hours), the RL must be able to converge to the best policy in a much shorter period. The main challenge in designing RL-based protocols is how to reduce the learning time, thus being able to fast react in case of link quality variability, without adding significant overhead and loosing performance.

In this work we start developing a Q-learning-based protocol tailored to the considered system model. Then we introduce enhancements on the Q-learning algorithm, tailored to the considered problem and based on Virtual Experience and State-action aggregation, showing that the performance of the proposed algorithms outperform the basic Q-learning and an aggressive strategy always transmitting packets in the buffer in terms of learning time, energy consumption, and Packet Delivery Latency (PDL).

The rest of this paper is organized as follows: in Section 2 we define our system model. In Section 3 we cast the problem as Markov Decision Process and introduce the definitions required to devise the RL solutions. In Section 4 we describe the proposed protocols. In Section 5 we evaluate their performance by means of simulations, comparing them with the Q-learning based protocol. Finally in Section 6 we conclude the paper pointing at future research directions.

This work was partially supported by the EU FP7 project ICT-SUNRISE, under grant no. 611449.

WUWNET '15, October 22-24 2015, Washington DC, USA

2. SYSTEM MODEL

We consider an underwater communication link between a transmitter and a receiver. The transmitter generates traffic according to a Poisson process with average interarrival time λ and constant packet payload size K.

We model the channel as a Binary Symmetric Channel (BSC) with time-varying Bit Error Probability (BEP) and statistically independent bit errors.

We consider time as divided in epochs of duration T, and assume that during each epoch the BEP is constant, whereas it may vary across consecutive epochs. We indicate with t the epoch index and with $b[t]$ the time varying BEP. The only assumption we need for devising our proposed protocol is that $b[t]$ is time correlated, i.e., the value of the channel at a given time provides an indication of the value it takes at a different time, and that correlation vanishes as the difference of the two time instants increases.

The transmitter stores a queue of incoming packets into a buffer of finite size $\overline{M}$, and keeps a running estimate of the channel BEP, which we indicate with $\hat{b}[t]$. The running BEP estimate is obtained and updated according to a procedure described in the following. Particularly, the estimation is *not* performed in every epoch, but only in selected epochs resulting from the policy described in the following section, i.e., $\hat{b}[t]$ does not refer to the actual BEP state in epoch t but to the BEP value of a past epoch of age $\delta[t]$. In other words, $\hat{b}[t]$ is an estimate of $b(t-\delta[t])$. Clearly, $\delta[t]$ increases by 1 in each epoch in which the channel is not probed, whereas it is reset to zero when a BEP estimation is performed. We assume that there is a limit $T_{\max}$ to the age (measured in number of epochs) of the running estimate $\hat{b}[t]$, i.e., the transmitter must update the estimate at most $T_{\max}$ epochs after the last one, i.e., $\delta[t] \in \mathcal{T}$, with $\mathcal{T} = \{0, \ldots, T_{max}\}$.

The channel estimate $\hat{b}$ is obtained as follows: the transmitter sends a channel probing packet of n_p bits which is known to the receiver; the receiver counts the number of bit errors and sends back to the transmitter the ratio between such number of errors and n_p. This value is further quantized, mapping it to a value in the set $\mathcal{B} = \{b_1, \ldots, b_{n_b}\}$. This quantized BEP value is the one used by the transmitter to run our proposed scheduling protocol, i.e., $\hat{b}[t] \in \mathcal{B}$.

The transmitter can encode data onto packets selecting a FEC code from a set of available codes $\mathcal{C} = \{c_1, \ldots, c_F\}$. Each code in the set is characterized by a different coding rate, i.e., it adds more or less redundancy, thus allowing the transmitter to trade off energy and packet transmission time with error performance. We indicate with r_c and τ_c the number of redundancy bits and the transmission time of a packet encoded with FEC code $c \in \mathcal{C}$. Accordingly, the *coded* packet size when using code c is $K + r_c$.

In each epoch, the transmitter sends a certain number of packets (selected on the basis of the transmission scheduling policy devised in Section 3). The receiver acknowledges the received packets. If a packet is not acknowledged within a pre-defined time interval, the receiver re-enqueues it at the bottom of the buffer.

The goal of our proposed strategy is to let the transmitter "learn" a transmission scheduling policy by which it can decide, in each epoch, (i) if to execute a BEP estimation, (ii) if and how many packets to transmit during the epoch and (iii) with which FEC code. The goal of the policy is to obtain good performance in terms of energy consumption while keeping a limited packet delivery latency (PDL), defined as the time difference between the time a packet incomes to the transmitter and the time it is correctly received.

3. MDP PROBLEM FORMULATION

We formulate the optimal transmission scheduling problem as a discrete time Markov Decision Process (MDP). In the following subsection we describe the state space, the actions/decisions space, the state transition dynamics and the selected cost function.

3.1 State

We define the state $s[t]$ in epoch t as the tuple composed by the running channel estimate $\hat{b}[t]$, its age $\delta[t]$, and the number of packets in the buffer at the beginning of epoch t, which we indicate with $M[t]$, i.e. :

$$s[t] = (\hat{b}[t], M[t], \delta[t]), \text{with } \hat{b}[t] \in \mathcal{B}, M[t] \in \mathcal{M}, \delta[t] \in \mathcal{T}, \quad (1)$$

where $\mathcal{M} = \{0, \ldots, \overline{M}\}$.

3.2 Actions/Decisions

At the beginning of each epoch, the transmitter may decide among a set of possible actions: (i) Transmit m packets encoded with FEC code c (and hence packet size $K + r_c$), with $m \in \{0, \ldots, m_{\max,c}\}$, where $m_{\max,c} = \min(M_t, \lfloor T/\tau_c \rfloor)$ is the maximum number of packets encoded with code c which can be transmitted in an epoch, or (ii) Transmit a channel probing packet of fixed size n_p to obtain a new BEP estimate $\hat{b}$.

More formally, for a state $s = (\hat{b}, M, \delta)$ the set of available actions $\mathcal{A}(s)$ can be defined as:

$$\mathcal{A}(s) = \begin{cases} \{a_p\}, & \text{if } \delta = T_{\max} \\ \{a_p\} \cup \{a_{m,c} | m \in \{0, \ldots, m_{\max,c}\}, c \in \mathcal{C}\}, & \\ & \text{otherwise.} \end{cases} \quad (2)$$

3.3 Transitions

State transitions are determined by the action taken and the packet arrival process. Given the current state $s = (\hat{b}, M, \delta)$ and an action a, the next state $s' = (\hat{b}', M', \delta')$ is determined as follows:

- $\hat{b}' = \begin{cases} \hat{b}_{\text{new}} & \text{if } a = a_p \\ \hat{b} & \text{otherwise} \end{cases}$, where $\hat{b}_{\text{new}}$ is the new observed BEP;
- $M' = M - m_s + l$, where $0 \le m_s \le m$ is the number of successfully transmitted packets (out of the m transmitted) and l the number of new arrived packets;
- $\delta' = \begin{cases} 0 & \text{if } a = a_p \\ \delta + 1 & \text{otherwise} \end{cases}$.

3.4 Costs

To each pair state-action (s, a) we associate a cost function $c(s, a)$, defined as a weighted function of costs associated to the delay and the energy consumption. The value of the cost function is computed at the end of each epoch and associated to the state-action pair of the epoch.

We propose to use the following cost function:

$$c(s,a) = \begin{cases} w_1 c_f(s,a) + w_2 c_o(s,a) + w_3 c_d(s,a), & a = a_{m,c} \\ c_{\text{msr}}, & a = a_p \end{cases} \quad (3)$$

where $c_f(s,a) = m_f/m$ is the cost associated to packet transmission failures, $c_o(s,a) = \frac{r_c - \min_{c'} r_{c'}}{\max_{c'} r_{c'} - \min_{c'} r_{c'}}$ the cost associated to the code overhead, $c_d(s,a) = (M_t - m_s)/\overline{M}$

the cost associated to packet delivery latency[1]. We observe that, as a by-product, since $c_d(s,a)$ tends to reduce the buffer occupation, it also allows to reduce the packet dropping probability, as elicited in the experimental evaluation. Finally, $c_{\text{msr}} = n_p/(K+\overline{r}_c)/\overline{M}$ is the cost associated to the transmission of the channel probing packet[2], where $\overline{r}_c$ is the number of redundancy bits added *on average* by the available codes (i.e. $\overline{r}_c = \frac{1}{F}\sum_{c=1}^{F} r_c$). The weights w_1, w_2, w_3 satisfy $w_1 + w_2 + w_3 = 1$ and each cost component is normalized in the closed interval $[0,1]$.

4. RL-BASED SOLUTIONS

4.1 Preliminaries

A policy is a function $\pi(s)$ which associates to each state s an action a, which in our problem corresponds to compute how many packets to send in each epoch, select the FEC code with which packets are error-protected, and whether to send a BEP probing packet. According to our MDP formulation, we are interested in determining the policy which minimizes the expected discounted cost over an infinite horizon with discounting factor $0 \leq \gamma < 1$, defined as:

$$V^{\pi}(s) = E_s^{\pi}\left\{\sum_{t=0}^{\infty}\gamma^t c(s_t,a_t)\Big| s_0 = s\right\}, \qquad (4)$$

where $c(s,a)$ is the expected immediate cost associated to state s and decision a. The discount factor models the preference for immediate costs. In particular, γ close to 1 leads to "far-sighted" evaluation while γ close to 0 leads to "myopic" evaluation, in which immediate cost minimization is strongly preferred.

An MDP can be solved via standard techniques, e.g., value iteration, LP formulation, etc [10]. However, computing the optimal policy is computationally expensive and requires a full a priori knowledge of system parameters such as the transition probabilities and the value of the cost functions for each-state action pair. This type of information, in practice, can hardly be assumed to be available *a priori*. Hence, this approach is unfeasible for the considered scenario. This motivates us to seek for RL-based techniques. These techniques are powerful because they are model-free techniques which *learn* the cost value function by experience, without requiring any knowledge of the transition probabilities.

4.2 Q-Learning

The Q-Learning algorithm is a simple algorithm that can be used to learn the optimal policy. It directly estimates the action-value function $Q^{\pi^*}(s,a)$ without any a priori knowledge about the system parameters. The knowledge of the Q-function is fundamental since it directly results into the associated policy. Indeed, the optimal policy can be simply computed as $\pi^*(s) = \arg\min_{a\in A(s)} Q^{\pi^*}(s,a),\ \forall s \in \mathcal{S}$ [10].

The core of the Q-Learning is a simple update step performed when the system transits from state s_t to state s_{t+1}. It is based on the *experience tuple* $\sigma_t = (s[t], a[t], c[t], s_[t+1])$, where $a[t]$ and $c[t]$ are the performed action and the cost per time unit when the system was in state $s[t]$. The update step is defined as follows [5]:

$$Q^{t+1}(s[t],a[t]) = (1-\alpha^t)Q^t(s[t],a[t]) + \alpha^t\left(c(s[t],a[t]) + \gamma \min_{a'} Q^t(s[t+1],a')\right) \qquad (5)$$

where a' is the *greedy* action in state $s[t+1]$, i.e., the action which minimizes the current estimate of the action-value function; $\alpha^t \in [0,1]$ is a time-varying learning rate parameter; and, $Q^0(s,a)$ can be initialized arbitrarily for all $(s,a) \in \mathcal{S} \times A$.

Q-Learning uses a sample average of the action value function to approximate $Q^{\pi^*}(s,a)$. Particularly, $Q^t(s,a)$ converges to $Q^{\pi^*}(s,a)$ with probability 1, for $i \to \infty$, if:

- α^t satisfies the stochastic approximation conditions $\sum_{t=0}^{\infty}\alpha^t = \infty$ and $\sum_{t=0}^{\infty}(\alpha^t)^2 = 0$;
- the instantaneous cost and transition probability functions are stationary;
- all of the state-action pairs are visited infinitely often.

Using Q-Learning, it is not obvious what is the best action to take in each state during the learning process. At any time the player can select the greedy action (the action with the lowest estimated action-value function $Q^t(s,a)$). When the player chooses the greedy action, he *exploits* the experience information base. If the player selects one of the non-greedy actions, he is *exploring*, i.e., he is improving the running estimates of the non-greedy actions-value functions $Q^t(s,a)$. Exploitation is the best way to minimize the expected cost on a single round, while exploration lets the player refine his experience information base and achieve a lower cost on the long run. Learning the best actions while at the same time minimizing the sum of paid costs requires to carefully trade-off between exploitation and exploration.

A common approach to trade-off between exploitation and exploration is to use the ϵ-greedy heuristic. The idea is to switch, every once in a while, from selecting the greedy action by performing one of the non-greedy ones. More formally, the player acts greedily with probability $1-\epsilon$ and non-greedily with probability ϵ, choosing the non-greedy action randomly from a uniform distribution. On the long run, this allows to update the estimated cost for all actions and re-think wrong approaches that seemed effective. We choose, among others, an ϵ-greedy strategy because, despite its simplicity, it still represents the best in terms of practically useful algorithms.

The major drawback of the Q-learning algorithm is that it only updates one state-action couple at a time resulting in low convergence rate and poor runtime performances, especially when the cardinality of the state space is large and there are many actions for each state. In the following we propose two methods for improving the convergence rate. The two methods are based on Virtual Experience and State-Action Aggregation [10].

4.3 Virtual Experience

The Virtual Experience method (VE) allows to update *multiple* state-action couples in each decision epoch in order to improve learning and runtime performances. The basic idea is that, whenever we send a given number of packets,

[1]Latency is inherent in this expression in the form of the number of packets that are present in the buffer at the beginning of epoch t and are *not* delivered, either because they are not sent in the epoch or because their transmission fails at the receiver. In any case, all such packets will incur an additional delay of one epoch. This number is then normalized to the maximum theoretical value it can achieve, i.e. $\overline{M}$ (all packets in a full buffer successfully transmitted).

[2]For consistency with $c_d(s,a)$, c_{msr} is defined as the number of data packets corresponding to the probing packet length n_p. However, since the length of data packets is not uniquely defined (it depends on the chosen FEC), we have used the average amount of redundancy $\overline{r}$.

we can use the information about the number of packets correctly received to emulate the outcome of different actions in possibly different states. Particularly, we use

$$\mathrm{PER}_{r_c} = m_f/m. \quad (6)$$

as an estimation of the PER at epoch t. The knowledge of PER_{r_c} allows us to perform *virtual experience*, i.e., to compute the outcomes m'_s and m'_f of multiple state-action couple (s', a'), as if these actions were actually performed. This information, along with the number l of newly arrived packets, is all we need to compute the virtual cost $c(s'[t], a'[t])$ and the virtual next state s'_{t+1} (see 3) and to update the relative entry of the Q-function using the Q-learning algorithm. In particular, we can perform virtual experience for each state-action couple (s', a') so that:

- the last measured BEP $\hat{b}'$ and the number of decision epochs since the last BEP probing δ' equal the ones in state s, i.e., only the buffer length differs;
- the amount of redundancy bits specified by a' is so that $r'_c = r_c$.

These are essential features to ensure that the BEP dynamics is consistently taken into account. More formally, we can compute m'_s and m'_f as follows:

- $\forall s' \in \mathcal{S} : \hat{b}' = \hat{b}$ and $\delta' = \delta$, $\forall a' \in \mathcal{A}(s') : r'_c = r_c \rightarrow m'_f = \mathrm{PER}_{r_c} m'$, $m'_s = m' - m'_f$.

The same arguments can be used to perform virtual experience also in case of the test action a_p. In that case, the outcome of the measurement and the number of packets arrived in the meantime are all is needed to compute $c(s'[t], a'[t])$ and $s'[t+1]$, and to update the value of the Q-function.

4.4 State-Action aggregation

The State-Action Aggregation method (S-AA) is based on the idea that reducing the cardinality of both the state space and the action set helps speeding-up the learning process. This is a challenging task, because there is a trade-off between the state space cardinality and the amount of information stored in the model, and a coarser grain model could negatively affect system runtime performances. The approach we propose in this work is simple and effective.

As a first step, we reduce the number of actions available in each state. Instead of allowing to transmit any possible amount of packets, we allow only the possibility to transmit, during an epoch, a pre-defined fraction of the number of packets in the buffer, i.e., a quarter, a third, half the buffer length, and so on. An option of particular interest, due to its simplicity, is the one in which the only possibility is to send all the packets in the buffer. For a state $s = (\hat{b}, M, \delta)$, the action space is hence given by:

$$\mathcal{A}(s) = \begin{cases} \{a_p\} & \text{if } \delta = T_{max} \\ \{a_p\} \cup \{a_{M,c} \mid c \in \mathcal{C}\} & \text{otherwise.} \end{cases} \quad (7)$$

As we can see, the only choice regards the FEC code, i.e., the amount of packet redundancy.

The second and equally important step is to reduce the cardinality of the state space. Since learning the BEP dynamics is a key element to minimize energy consumption, we keep the BEP and the BEP estimate age state components ($\hat{b}$ and δ) untouched, and reduce the state space cardinality acting on the buffer component. We propose a hard aggregation approach in which several states s_m corresponding to different buffer lengths m (but with the same value of $\hat{b}$ and δ) are aggregated in a single state. In practice, the buffer is divided in ξ slots[3] of equal length $\beta = \lceil \overline{M}/\xi \rceil$ and each state s_m is mapped to slot $i = \lceil m/\beta \rceil$, to form an aggregated state s_i. In the next section we will show that we can achieve very good results simply setting $\xi = 1$.

5. PERFORMANCE EVALUATION

In this Section we evaluate and compare the performance of the two proposed techniques and the Q-learning based one. The selected set of error correction codes is generated starting with the rate 1/2 convolutional code used in the Janus standard [9] and then applying puncturing to obtain a rate 2/3. To generate the channel trace we used a discrete-time Markov Chain[4] with BEP values in the interval $[10^{-3}, 10^{-1}]$. The other system parameters are listed in the following table:

Parameter	symbol	value/range
Buffer size	$M_{\max}$	100 packets
Maximum number of packets that can be transmitted in an epoch	$m_{\max}$	100 packets
BEP probing packet size	n_p	1000 bits
Payload size	K	400 bits
Redundancy bits	$\frac{K}{K+r_c}$	$\frac{1}{2}, \frac{3}{4}$
Maximum time to refresh BEP estimate	$T_{\max}$	15 epochs
Incoming packets arrival rate	λ	2 packets/epoch
Epoch duration	T	50 seconds
Discount factor	γ	0,9
Aggregation factor	ξ	1
Bit rate		1200 bps
Transimt power		3,3 Watts

The selected bit rate and transmit power values are set to values common in commercial modems.

Fig. 1 shows the evolution of the cost function during the simulation time. This kind of plots are useful in assessing the performance of a learning method from the point of view of (i) effectiveness in achieving a policy that has good performance in terms of cost minimisation, and (ii) doing it in a reasonable time, so that the technique is able to adapt to a variation in the long-term channel behavior without loosing performance due excessively long transient regimes. The plots show the average of the cost evolution over ten realizations with different initial conditions. It can be seen that QL takes a time longer than the simulation length to converge to the optimal policy. Conversely, both S-AA and VE guarantee a fast convergence to their best selected policy. With our simulation setup, S-AA achieves the minimum value of the cost function, although VE achieves a close value. The important aspects of these performance are that (i) both the proposed techniques achieve a close to optimal value of the experienced cost; (ii) they converge order of magnitudes faster than Q-learning, which makes them suitable for use in scenarios where the environment changes on a time scale in the order of hours; (iii) although S-AA has a coarser state-action space, it performs slightly better than VE. This reflects a typical tradeoff, where learning on a coarser state-action space can yield better performance

[3]We assume $\overline{M}$ to be an integer multiple of ξ.

[4]Note that this is only done to have a trace over which the algorithms can run, there is no loss of generality in this choice.

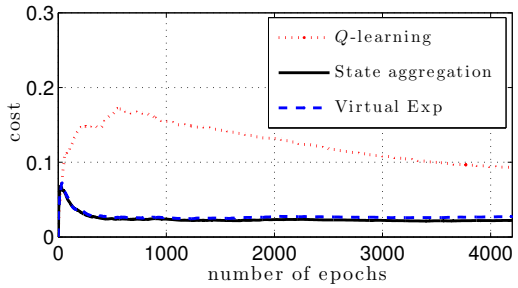

Figure 1: Cost evolution over time.

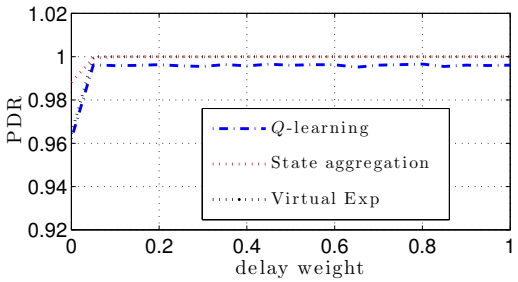

Figure 2: Packet Delivery Rate.

than learning over a finer state-action space, which requires to explore more state-action pairs. Finally, it is worth mentioning that both techniques entail a limited average value of the energy used for channel probing, corresponding to 12% of the total energy.

Fig. 2 shows the packet delivery ratio as a function of the weight assigned to delay in the cost function. VE and S-AA virtually guarantee the delivery of all packets for all the weights, except for the case in which the delay is assigned a zero weight. However, QL drops around 1% of the packets. This is due to the fact that the learning rate is so slow that the optimal policy is not achieved during the simulation runs: the transmitter doesn't learn the way to stabilize the buffer, i.e. to keep latency limited.

Fig. 3 shows bidimensional performance planes in which vertical axis represent PDL and horizontal axix energy consumption. Each point of the scattered plots represents a distinct value of the delay weight in the cost function. VE and S-AA outperform QL in terms of delay performance, delivering packets within 50 seconds or faster. Their energy vs. latency performance also well matches the application requirements: as the weights associated to the different costs change the protocol behavior also changes, compromising between energy and latency as requested. The figures also show the performance of an aggressive policy where the transmitter always attempts to transmit all the packets in the buffer, using always the rate 1/2 code and the rate 3/4 code, and without any channel probing. The RL-based techniques result in comparable latency but significantly lower energy consumption, despite the energy used for channel probing.

6. CONCLUSION

We have introduced two RL-based techniques for UWSNs Data-Link packets transmission scheduling and channel probing. The results of our work show that RL is a promising tool for optimizing the performance of Data-Link protocols in scenarios characterized by uncertain time-variability of the transmission conditions.

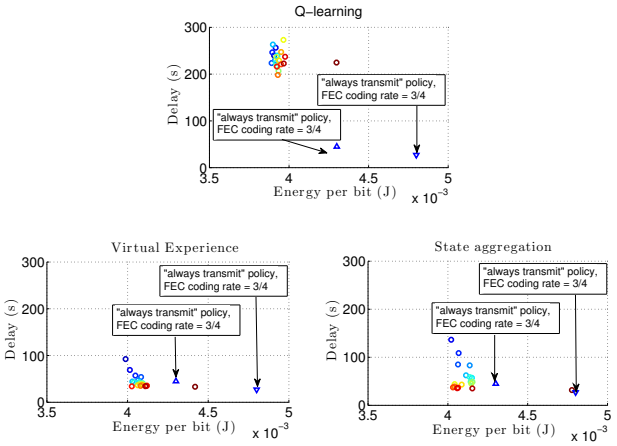

Figure 3: Energy consumption and PDL.

7. REFERENCES

[1] M. Chitre and K. Pelekanakis. "Channel variability measurements in an underwater acoustic network." Underwater Communications and Networking (UComms), 2014. IEEE, 2014.

[2] P. Casari, M. Rossi, M. Zorzi, Towards optimal broadcasting policies for HARQ based on fountain codes in underwater networks, in: Proc. IEEE/IFIP WONS, 2008.

[3] . Tomasi et al., Redundancy allocation in time-varying channels with long propagation delays, Ad Hoc Netw. (2015), http://dx.doi.org/10.1016/j.adhoc.2015.01.009

[4] B. Tomasi et al., Cross-layer analysis via Markov models of incremental redundancy hybrid ARQ over underwater acoustic channels, in press, Ad Hoc Netw. (2014), http://dx.doi.org/10.1016/j.adhoc.2014.07.013

[5] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. Cambridge: MIT press, 1998.

[6] N. Mastronarde and M. van der Schaar. "Joint physical-layer and system-level power management for delay-sensitive wireless communications." IEEE Trans. Mob. Comp. 12.4 (2013): 694-709.

[7] T. Hu and Y. Fei, QELAR: A Machine-Learning-Based Adaptive Routing Protocol for Energy-Efficient and Lifetime-Extended Underwater Sensor Networks, IEEE Trans. Mob. Comp, Vol. 9, No. 6, pp. 796–808, JUNE 2010.

[8] R. Plate, C. Wakayama, Utilizing kinematics and selective sweeping in reinforcement learning-based routing algorithms for underwater networks, in press, Ad Hoc Netw. (2014), http://dx.doi.org/10.1016/j.adhoc.2014.09.012.

[9] B. Tomasi, et al. "On modeling JANUS packet errors over a shallow water acoustic channel using Markov and hidden Markov models." Proc. 2010 IEEE Military Communications Conference (MILCOM 2010), San Jose, Ca, USA, Oct. 31 - Nov. 3, 2010.

[10] M. L. Puterman, "Markov decision processes: discrete stochastic dynamic programming", Wiley, 2019.