Tuning an underwater communication link

Satish Shankar
Acoustic Research Lab
National University of Singapore
mailshanx@gmail.com

Mandar Chitre
Acoustic Research Lab
National University of Singapore
mandar@arl.nus.edu.sg

***Abstract*—We present machine learning algorithms to tune an underwater communication link. The link tuner is characterized by 3 features: a) It is data driven, rather than physics driven. Hence, it only needs bit error rate information as input and is independent of the modem implementation, b) The tuner balances exploration of the search space against exploitation of existing knowledge, and c) It optimizes for the average data rate, instead of searching for maximum possible data rate. We implement the link tuner on the UNET-II modem and present results from simulations, water tank tests and field trials. The results demonstrate a significant improvement in average data rate as compared to the average data rate attained without tuning.**

I. Introduction

For any given modem implementation, the data rate performance of a link depends on the parameters of the communication algorithms in use. Determining what might be a good set of parameter values to use in a given channel is critical for achieving high data rates.

The motivation for a link tuner arises from:

1) The large number of possible configurations for a link.
2) The need for an online tuning mechanism.

For modems with a variety of configurable parameters, the space of parameter values can quickly grow to be large. As an illustrative example, consider the UNET-II modem developed at the Acoustic Research Lab (ARL) [1]. The modulation system is based on OFDM. It is possible to set values for the following parameters:

1) Modulation mode (coherent / incoherent and QPSK / BPSK).
2) Differential mode (time or frequency).
3) Prefix / suffix length (values range from 0 to 1024)
4) Number of carriers (64, 128, 256, 512, 1024)
5) Number of nulls (values range from 0 to 1024)
6) Error correction type (Convolution, Golay, both or none)

Together, they represent 10's of millions of possible configurations.

The need for an online tuning mechanism is evident by considering the dynamic nature of the underwater channel. There is no such thing as a "typical" underwater channel [2]. The channel response changes with location, water depth, temperature/salinity profiles, modulation methods used, frequency of operation, tidal cycles, and a myriad of other factors. That makes it impossible to search for the best set of parameter values offline and store them. Hence we need a mechanism for online tuning, so that we can continuously make improvements to the average data rate in response to any channel.

A. Data driven link tuning

We adopt a data driven approach to link tuning, rather than a physics driven one. Specifically, we make a series of choices for parameter values based only on bit error rate (BER) data. We chose a data driven approach because of two advantages:

- Simplicity in terms of input requirements to the tuner: Physics based models are based on information such as absorption/spreading loss [2], ray/beam tracing information [3], Doppler / delay spread [4], and so on. In contrast, the input to our data driven link tuner simply consists of bit error rate data. BER information is a fundamental quantity in a communication system, hence one may safely assume its availability.
- Decoupling of the tuner from the modem implementation: A single data driven link tuning framework can operate on different modems and it remains useful even as the underlying physical layer implementation evolves and changes.

B. Exploration vs exploitation

In order to make better choices in the future, the tuner needs to try new actions by exploring the action space. But in order to generate more reward, the tuner needs to exploit its existing knowledge. This leads us to a central theme: that of balancing exploration of the search space in order to make better decisions in the future versus exploitation of the current knowledge to increase data rate performance now.

Pursuing a purely exploration or exploitation based policy leads to poor performance, so we need to find some sweet spot in the middle. Balancing exploration and exploitation is critical for achieving high average data rates.

C. Optimizing for average data rate vs maximum data rate

We are concerned with the whole problem of maximizing the *average* data rate, rather than attaining the maximum possible data rate. We measure the average over the course of a data transfer. The time taken to tune the link is included while measuring the average data rate.

It may be possible to devise a search algorithm to search for the configuration with the maximum possible data rate. We can run the search process until either we have examined all configurations, or we meet a stopping criterion such as a timeout or we have discovered a configuration with data rates higher than a pre-set threshold.

Rather than decide on an arbitrary stopping criterion, we aim to maximize the average data rate. In other words, we aim to optimize the entire sequence of parameter configurations that are tried out in the process of link tuning.

II. Literature Review

The ideas explored in this paper lie in the intersection of the fields of adaptive modulation and machine learning. We first review literature that provides an overview of these fields. We then review some specific literature that has close parallels to our work. Finally we explain the key differences in our work and how we build on existing literature.

A. Adaptive modulation

See [5], [6] and [7] for a sampling of literature from the field of adaptive modulation, particularly in the cognitive radio domain. The adaptation schemes often make use of SNR information and in the case of MIMO-OFDM systems, of spatial diversity / beamforming [7]. See [8], [9] and [10] for examples of research effort directed towards channel estimation algorithms in both terrestrial wireless networks as well as the underwater communication domain. The authors in [11] develop a stochastic learning automaton for adapting an ODFM link by comparing BER estimates for various schemes with associated threshold values.

B. Machine learning methods

A distinctive feature of our work is the application of machine learning methods to digital communication. In 1959, Arthur Samuel defined machine learning as a ”Field of study that gives computers the ability to learn without being explicitly programmed”. See [12] for a comprehensive overview of this field.

The space of machine learning algorithms can be divided into 3 primary partitions: a) supervised learning, b) unsupervised learning and c) reinforcement learning.

Reinforcement learning is concerned with learning by interaction [13]. Specifically, an *agent* seeks to take *actions* in an *environment* to maximize its cumulative reward. The agent is not presented a set of training examples with labels: instead, the agent needs to generate its own set of training data by interacting with the environment. The focus in this setting lies in the agent’s online performance. There is a critical need to balance between an exploration of uncharted space and the exploitation of existing knowledge.

We use the multi armed bandit problem [14] as our model of choice for studying the tradeoff between exploration and exploitation. The Gittins index [15] provides an optimal policy for maximizing the expected discounted reward is certain assumptions on reward distributions hold. We also point to the upper confidence bounds (UCB) family of algorithms [16] because they are regarded as the state of the art algorithms in a classical multi armed bandit setting [17].

C. Closely related work

The most closely related work in terms of applying reinforcement learning methods to a communication link is done by Haris et. al in [18], [19] and [20]. The authors try many exploration heuristics such as Boltzmann, ϵ-greedy and Gittins indexing [15] methods (among others) to optimize the bit rate per hertz of a cognitive radio.

We also point out [21] and [22], which are the only literature we could find that discuss reinforcement learning ideas applied to an underwater communication link. The authors try exploration heuristics similar to Boltzmann and ϵ-greedy methods, and also develop other custom strategies.

Our work builds on the above literature. A distinctive feature of our work is that we address 3 key aspects of the tuning problem which are not considered above. These aspects turned out to be critical to achieving a performant tuner:

1) Finite vs. infinite search space:
The above literature implicitly assumes that the search space for a tuner is finite. In practice, the set of candidate search points under consideration by the tuner would need to be reasonably small in order to meet memory and time limitations for implementation on a practical system. We present here a technique for dynamically generating new candidate search points in response to past results. This allows us to consider a very large or even an infinite search space.
2) Estimation of packet success rates:
We use a Kalman-filtering approach to fuse information from user data packets along with training packets to get bit error rate estimates. We then extrapolate this knowledge to obtain final reward estimates. This approach better utilizes the incoming stream of packet information to derive a robust estimator, compared to simply counting the number of successful / failed packet transmissions and estimating a packet success rate from that.
3) Arbitrary reward distribution:
Some of the more sophisticated techniques examined in the above works (such as the Gittins index [15]) require that all actions yield the same reward upon success. This renders the Gittins index unusable in a link tuning scenario as described here, because actions do not yield equal rewards. As an alternative to the Gittins index, we derive a dynamic programming based index that can be used with arbitrary reward distributions.

III. Problem definition

We define a *scheme* $s \in \mathbb{S}$ to be a vector that contains the values of each parameter we wish to tune. s_i^{p} denotes a packet encoded in scheme i. We can apply an encoding $c \in \mathbb{C}$ to a packet, yielding an information rate of r_c. The information rate is defined as the the ratio of the user data carrying bits to the total bits in the packet. We have 2 types of packets, depending on the encoding c applied to them: data packets and test packets.

The encoding applied to a data packet is an error correction code. Since error correction coding is also popularly known as forward error correction (FEC) in literature, we shall henceforth use both terms interchangeably. A data packet has an information rate r_c that is equal to the code rate of the FEC it is encoded with.

Test packets do not use any error correction coding. Test packets are pre-generated bit patterns, useful for BER estimation. A test packet has an information rate of zero. A modem can receive a test packet n bits in length and can compute the number of bits k that were erroneously received.

Based on the above descriptions of data and test packets, we note that the general notions of an encoding c with information rate r_c applies to all packets. However in the context of data packets, these correspond to an error coding scheme c with a code rate r_c.

We define the uncoded data rate of a scheme β_j to be equal to the data rate that one would obtain by setting $r_c = 1$. β_j corresponds to an under-specified scheme (it does not specify the encoding c). For a data packet, it is equal to the data rate that would result by not employing any FEC.

For a given β_j, we arrive at scheme i (a complete specification) if we also specify the information rate r_c. Note the correspondence between $i \equiv (j, c)$. A packet s_i^{p} encoded in scheme i is equivalently encoded in $s_{j,c}^{\mathrm{p}}$. Upon successful reception, s_i^{p} yields an instantaneous data rate of $\beta_j r_c$.

Each error coding scheme c can tolerate an uncoded bit error rate upto a certain threshold T_c. We compute the value of T_c depending on the FEC used. For example, a $[24, 12, 8]$ Golay code is a block code that carries 12 bits of user information in a 24 bit code word. It is capable of correcting upto 3 error bits. If we assume that bit errors are independently distributed, a bit stream encoded with this FEC will not tolerate an average bit error rate greater than $3/24 = 12.5\%$. In general the lower the code rate, greater the threshold tolerance T_c.

We have a point to point communication link that can return feedback on our scheme choices. For test packets, the receiver reports the number of erroneous bits k. For a data packet, the receiver reports successful / failed reception. We are free to switch to any scheme at every transmission opportunity. The modems use a control scheme for communicating control information such as as number of erroneous bits or packet reception success / failure. The control scheme is a robust, low data rate scheme. It does not get tuned and is used only for communicating control information.

Given the above setup, we seek to compute a sequence of schemes to maximize the average data rate.

IV. Architectural overview

We use the multi armed bandit (MAB) problem as a model for the link tuning process [14]. Suppose one is presented with a set of slot machines. When played, a machine yields some reward as per an unknown probability distribution. The multi armed bandit problem is defined as follows: in what sequence must one play the machines to maximize the total expected reward?

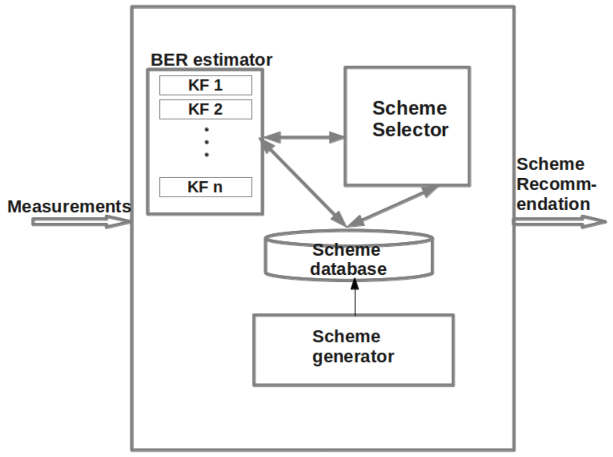

Fig. 1. Link tuner architecture

The mapping between the MAB problem and the link tuning process is as follows: a bandit arm in the MAB problem is defined to be the same as a scheme, except that the encoding c is left unspecified. A bandit arm j together with an encoding c defines a scheme i. Thus each bandit arm can be associated with multiple schemes, depending on the choice of c. A packet s_i^{p} corresponds to scheme i, bandit arm j, and an encoding of information rate r_c. Note the correspondence between the indices $i \equiv (j, c)$. The uncoded data rate of s_i^{p} is β_j. If s_i^{p} is successfully received, it yields an instantaneous data rate of $\beta_j r_c$.

Fig. 1 shows a high level overview of the link tuner architecture. The input to the link tuner consist of measurements derived from BER information. The output of the link tuner consists of recommendations for which scheme to use for the next transmission. The link tuner itself consists of 4 components: a database of schemes, a scheme generator, a BER estimator, and a scheme selector. The BER estimator maintains an estimate a BER estimate for each scheme in the database. The scheme generator periodically samples new schemes and adds them to the database. The scheme selector uses bandit selection algorithms to provide a scheme recommendation.

V. BER Estimation and Scheme Generation

The BER estimator is implemented as a set of Kalman filters, such that each bandit arm has an associated Kalman filter.

A. *Kalman filter for bandit arm*

Fig. 2 shows an overview of the Kalman filter. The filter maintains a state estimate $\hat{x}$, which is a function of the BER. We maintain a filter for each bandit arm j. The estimate undergoes a time update after every transmit-receive cycle. The time update represents the decrease in confidence of an estimate over time due to a potentially time varying channel. When a packet is received, we compute a measurement and perform

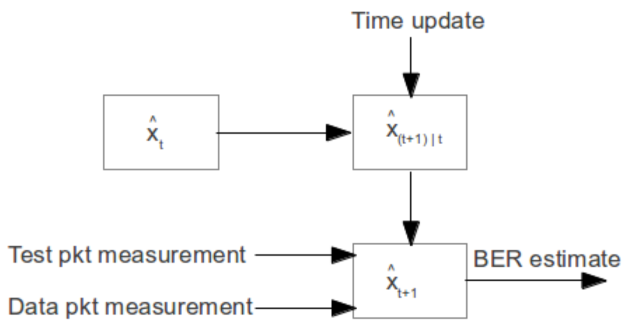

Fig. 2. Kalman filter for BER estimation

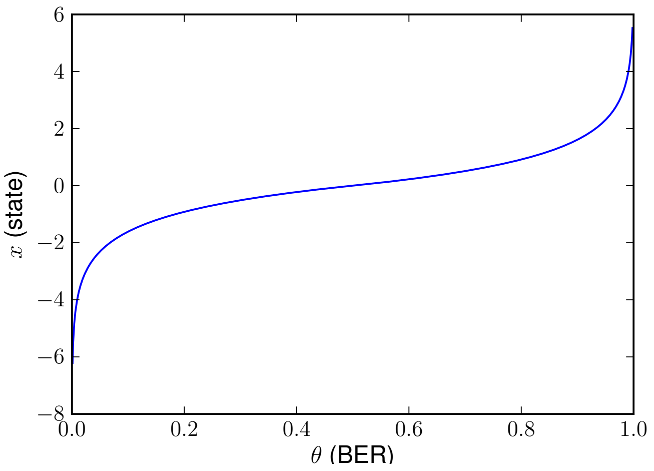

Fig. 3. Nonlinear mapping from BER to state

a measurement update for the corresponding filter. The time update is applied to every filter, whereas the measurement update only applies to the arm j corresponding to the received packet.

Let $0 \leq \theta \leq 0.5$ be a constant, but unknown, BER associated with a static channel. We observe one of two measurements associated with the BER:

- Test packet measurement: A test packet with n bits is received with k erroneous bits.
- Data packet measurement: A coded data packet with n coded bits is received successfully or unsuccessfully. If it is received successfully, we assume that the number of erroneous bits $0 \leq k \leq nTc$ where T_c is a BER threshold for FEC c. If it was received unsuccessfully, we assume that $nT_c \leq k \leq n/2$.

We wish to find an online estimator for θ given a stream of incoming measurements.

1) Model: Typically BER values are small, of the order of less than 1%. The variation in BER values is smaller still, of the order of much less than 1%. We need a mapping function $f : \theta \rightarrow x$ that would be highly sensitive to θ at these values, where x is the state. Based on this requirement, we use the following non-linear mapping $f(\cdot)$ of BER θ as a state variable:

$$x = f(\theta) = \begin{cases} \log(2\theta) & \theta \leq 0.5 \\ -\log[2(1-\theta)] & \theta > 0.5 \end{cases} \tag{1}$$

Fig. 3 shows the mapping from θ to x.

The system model is given by:

$$x_t = x_{t-1} + w_t \tag{2}$$

where $w \sim N(0, Q)$ is the process noise.

When a test packet measurement is available, the measurement $z_t = f(k/n)$ is a noisy measurement of the state x_t such that:

$$z_t = x_t + v_t \tag{3}$$

where measurement noise $v_t \sim N(0, \sigma_t^2)$ and

$$\sigma_t^2 = \frac{k}{n^2}(1 - \frac{k}{n})f'(\frac{k}{n}) \tag{4}$$

as k follows a Binomial distribution. Here $f'(\cdot)$ is the derivative of $f(\cdot)$. When a data packet success or failure is available, the measurement is given by;

$$z_t = \begin{cases} f(\frac{T_c}{2}) & \text{success} \\ f(\frac{T_c}{2} + \frac{1}{4}) & \text{failure} \end{cases} \tag{5}$$

As before, z_t is a noisy measurement such that:

$$z_t = x_t + v_t \tag{6}$$

where measurement noise $v_t \sim N(0, \sigma_t^2)$ and

$$\sigma_t^2 = \begin{cases} \frac{T_c^2}{12} f'(\frac{T_c}{2}) & \text{success} \\ \frac{(0.5-T_c)^2}{12} f'(\frac{T_c}{2} + \frac{1}{4}) & \text{failure} \end{cases} \tag{7}$$

2) Filter update equations: Given the above model, we have the following filter update equations where $\hat{x}$ is the estimated state with variance P. The Kalman equations can be found in [23]. However, [24] provides a much more accessible version of the equations, and we use those to derive ours.

The time update equations are given by:

$$\hat{x}_{t|t-1} = \hat{x}_{t-1} \tag{8}$$

$$P_{t|t-1} = P_{t-1} + Q \tag{9}$$

The measurement update equations are given by:

$$K_t = \frac{P_{t|t-1}}{P_{t|t-1} + \sigma_t^2} \tag{10}$$

$$\hat{x}_t = (1 - K_t)\hat{x}_{t|t-1} + K_t z_t \tag{11}$$

$$P_t = (1 - K_t)P_{t|t-1} \tag{12}$$

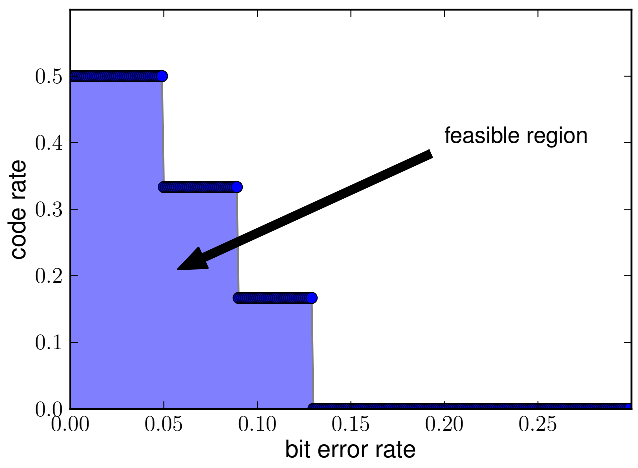

Fig. 4. Code rate vs BER

B. FEC selection logic

We utilize the bit error rate estimates to determine which FEC c would be appropriate to use for the given bandit arm. The relationship between code rate and BER for the error correction codes implemented in the UNET-II modem is illustrated in Fig. 4, which marks a feasible region. The bit errors in a packet received in the feasible region can be corrected for by the FEC. Packets received outside the feasible region have too many bit errors for the FEC to correct. For example, a packet encoded with an FEC with code rate 0.33 can be successfully received if it has a bit error rate of less than 7%.

Let β_j be the (uncoded) data rate of bandit arm j and $\hat{\theta}_j$ be its estimated bit error rate.We maintain a Kalman filter for each bandit arm to compute $\hat{\theta}$. Each encoding $c \in \mathbb{C}$ has a threshold bit error rate T_c and encodes a packet with an information rate r_c. Note that for test packets, $r_c = 0$ for all values of T_c. In fact there is no notion of T_c for a test packet, since the encoding c used in a test packet is not an FEC. The value of T_c can be read from Fig. 4. For example, an FEC with $r_c = 0.33$ has $T_c = 7\%$. The received packet can not be successfully decoded if the bit error rate exceeds T_c. Each arm j is associated with $|\mathbb{C}|$ encodings. Scheme s_i is composed of the parameters specified in arm j together with encoding c. A packet s_i^{p} encoded in scheme s_i has a packet success probability given by:

$$\hat{\alpha}_i = \hat{\alpha}_{j,c} = \begin{cases} 1 & \hat{\theta}_j < T_c \\ 0 & \hat{\theta}_j \geq T_c \end{cases} \tag{13}$$

$\hat{\mu}_i$ is the estimated data rate of scheme s_i and is given by:

$$\hat{\mu}_i = \hat{\mu}_{j,c} = \hat{\alpha}_i \beta_j r_c \tag{14}$$

C. Scheme generator

Since the parameter space is exceedingly large, it is impractical to enumerate all schemes and maintain indices/policies over all of them. The scheme generator overcomes this problem by progressively adding new schemes to the scheme database.

We have 3 requirements for the scheme generator:

- It needs to consider the parameter space in its entirety, without discarding any part of the search space.
- It must progressively sample schemes in the neighborhood of the more successful schemes.
- We need a mechanism for controlling the probability that a newly generated scheme lies close to the most successful schemes

Based on these requirements, we arrived at the following heuristics to generate new schemes:

- We draw schemes from a probability density function (PDF) curve, which we modify as time progresses. Fig. 5 illustrates the evolution of the PDF for an example parameter. Each parameter has its own PDF.
  We manipulate the shape of the PDF for each parameter independently, even though in reality the PDFs of the parameters are likely not independent. Doing so is acceptable because we do not know the relationship between the PDF's of parameters. Given the absence of any way to easily deduce these relationships, we treat the PDFs independently. That way, we retain the option to sample schemes in all directions.
- In the beginning, the schemes are drawn from a uniform distribution. This meets the first requirement, i.e. we consider the parameter space in its entirety. We draw d_1 schemes and store them in the database initially.
- After d_2 transmissions, new schemes are generated. We determine which scheme has the maximum expected data rate among the ones that were tried. We refer to this scheme as the *winning scheme*.
- We then add d_3 schemes to our database. These schemes are drawn from a normal distribution, centered at the winning scheme. Drawing schemes from a normal distribution allows us to meet the second and third requirements: we now have higher probability of sampling points closer to the previously successful schemes. Changing the standard deviation of the curve provides us with a mechanism to control the probability with which the newly generated schemes lie close to the previously successful ones.
- If all the previously tried schemes have an estimated data rate of zero, we draw again from the uniform distribution.
- Suppose one of the parameters can take values between 0 and 100, and the winning scheme had the value of this parameter = 75 We then generate a normal distribution with mean = 75. Also, 75 divides this range in two intervals: $[0, 74)$ and $[75, 100]$. We set the standard deviation of the normal curve to be half the range of the smaller interval, i.e. $(100 - 75)/2 = 12.5$. At every d_2 transmissions it is time to generate new schemes again. At this point, we multiply the standard deviation by a shrink factor $d_f \in (0, 1)$.

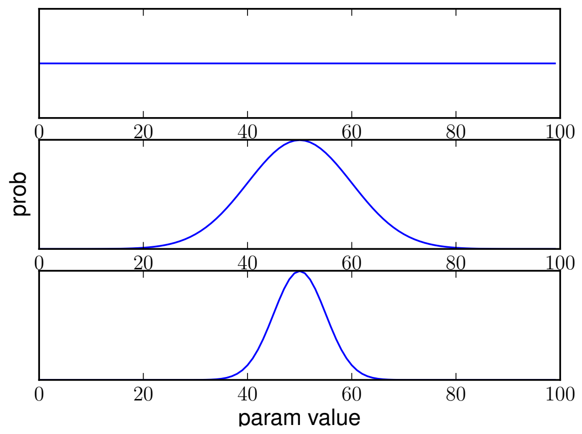

Fig. 5. Evolution of probability density curves used by scheme generator

We set $d_1 = 10, d_2 = 6, d_3 = 3$ and $d_f = 0.9$. The choice of using a normal distribution, the method we use for setting its standard deviation, as well as the values of d_1, d_2, d_3 and d_f are heuristics based on our experience:

- If d_2 is too large and none of the schemes in the database are performing well, then we have to wait have to wait for a long time before exploring any new schemes. If d_2 is too small, then we end up generating new schemes too fast.
- If d_3 is set to be much larger than d_2, then we end up accumulating too many schemes that never get tried. Hence, our choice of value for d_3 is primarily guided by memory constraints,
- The tuner performance is not very sensitive to d_1.
- d_f controls the rate at which we narrow width of our normal distribution, and we have found that a value of 0.9 works well in practice.
- Setting the standard deviation of the normal curve to be equal to half of the smaller interval between the mean and the extreme values of the parameter makes the curve more symmetric about the mean. It is still skewed due to the finite support, but we find that in practice it doesn't affect the performance significantly.

VI. Bandit Selection Algorithms

From the set of available schemes in the database, we need to select one for transmission. The selection logic is governed by bandit algorithms, which we explore several of.

A. Dynamic programming (DP) solution

A dynamic programming solution arises out out of modeling the link tuning process as a sequential decision optimization process. We explain this view of link tuning in more detail before defining the dynamic programming equations.

1) Link tuning as sequential decision optimization: The process of link tuning can be viewed as a sequential decision optimization problem. At any given time, we possess a certain amount of knowledge about our schemes and their performance in the channel. We can think of possessing this knowledge as equivalent to being in a certain state. Given this state, we need to make a decision and choose a scheme. Receiving the results of the transmission allows us to update our state, and then it's time to make the decision again. Thus the link tuning process is essentially a decision process. We wish to optimize our decisions to maximize the average data rate. Note that maximizing the average data rate requires us to optimize the entire series of decisions, rather than any particular decision in isolation.

Dynamic programming is an algorithmic design paradigm that is well suited to solve problems of this nature. Consider an environment where we can transition from state ξ_t at time t to $T(\xi_t, a) = \xi_{t+1}$ by choosing an action a. $T(\xi_t, a)$ is a function that describes the state transitions. Choosing action a yields an immediate reward. The expected value of immediate rewards are given by a reward function $R(\xi_t, a)$.

The key idea of dynamic programming lies in computing the *value function* $V(\xi_t)$. The value function is a measure of the inherent goodness of that state. The optimal value function $V^*(\xi_t)$ describes the maximum total expected reward looking into the future:

$$V^*(\xi) = \arg\max_a \{R(\xi_t, a) + V^*(T(\xi_t, a))\} \tag{15}$$

Equation (15) is known as the Bellman optimality equation in literature [25]. Note that we do not have an explicit expression for $T(\xi, a)$ in the context of the link tuner, and instead compute it implicitly.

2) Dynamic programming equations: Let the data rate, packet success probability estimate and bit error rate estimate for scheme i (which is associated with arm j and error coding scheme c) at current time t be $\hat{\mu}_{i,t}$, $\hat{\alpha}_{(j,c),t}$ and $\hat{\theta}_{j,t}$. Note the correspondence between i and (j, c). We define the state ξ_t to consist of the 3-tuple $(\hat{\alpha}_{(j,c),t}, \beta_j, r_c)$. Choosing an action corresponds to transmitting a packet s_i^{p}.The immediate expected reward is given by the function $R(\xi_t, s_i^{\mathrm{p}})$:

$$R(\xi_t, s_i^{\mathrm{p}}) = \hat{\alpha}_{(j,c),t}\beta_j r_c \tag{16}$$

Given the results of transmitting packet s_i^{p}, we use the equations in section V-A2 to compute the new state $\xi_{(t+1)|s_i^{\mathrm{p}}}$. To compute $\xi_{(t+1)|s_i^{\mathrm{p}}}$, we only need to update $\hat{\alpha}_{(j,c),t}$ since β_j and r_c don't change with time. To update $\hat{\alpha}_{(j,c),t}$, we first update $\hat{\theta}_j$ using the Kalman filter equations in section V-A2, followed by Equation (1) to update $\hat{\alpha}_{(j,c),t}$.

Given that we transmitted s_i^{p}, updating $\hat{\alpha}_{(j,c),t}$ is conditioned on a successful / failed reception of s_i^{p}. Since failed receptions yield zero reward, we only need to compute the update conditioned on successful reception, and multiply that by the probability of successfully receiving s_i^{p}. Putting all of this together, we arrive at the following expression for the optimal value function for scheme i in state ξ_t:

$$V_i^*(\xi_t) = \arg\max_{j,c} \left\{ R(\xi_t) + \sum_{c \in \mathbb{C}} \hat{\alpha}_{(j,c),t} V_i^* \left(\xi_{(t+1)|s_i^{\mathrm{p}}} \right) \right\} \tag{17}$$

We define the following policy based on the optimal value function. At each time step, we choose the scheme i that maximizes $V_i^*(\xi_t)$:

$$j(t) = \arg\max_i V_i^*(\xi_t) \tag{18}$$

B. Softmax Exploration

Let $\hat{\mu}_i(t)$ be the expected data rate of scheme i at time t. $p_i(t)$, the probability of choosing scheme i at time t is given by:

$$p_i(t+1) = \frac{e^{\hat{\mu}_i(t)/\tau}}{\sum_i e^{\hat{\mu}_i(t)/\tau}} \tag{19}$$

Where τ is a temperature parameter. When $\tau = 0$, the heuristic exhibits pure greedy behavior. As τ tends to infinity, the algorithm picks schemes uniformly at random. One could possibly use decreasing τ schedules, but [26] show empirical evidence that such schedules offer no practical advantages, so we choose a fixed value of $\tau = 0.1$ for our experiments.

C. ϵ-greedy

The ϵ - greedy is a commonly used heuristic in the AI community. It is very simple and well suited for use in sequential decision problems. Given its widespread use and adoption, we implement a version in the link tuner. The probability of choosing a scheme i at time t is given by:

$$p_i(t+1) = \begin{cases} 1 - \epsilon + \epsilon/k & \text{if i = } \arg\max_i \hat{\mu}_i(t) \\ \epsilon/k & \text{otherwise} \end{cases} \tag{20}$$

One could possibly vary ϵ adaptively. However, [26] presents empirical evidence that finds no practical advantage in doing so. In our experiments we find that a value of 0.05 works well, so we use that for our experiments.

D. Pursuit algorithm

Pursuit algorithms maintain a policy over each bandit arm, the updates to which depend on the estimated reward of the arm. We use a version of the pursuit algorithm as described in [13]. Each scheme is assigned a uniform probability of selection, $p_i(t) = 1/|\mathbb{S}|$. The probabilities are updated as per:

$$p_i(t+1) = \begin{cases} p_i(t) + \beta(1 - p_i(t)) & \text{if i = } \arg\max_i \hat{\mu}_i(t) \\ p_i(t) + \beta(0 - p_i(t)) & \text{otherwise} \end{cases} \tag{21}$$

where $\beta \in (0,1)$ is the learning rate. We have found that $\beta = 0.9$ generally works well, so we use that value for our experiments.

E. Upper Confidence Bounds (UCB)

The UCB family of algorithms are regarded as the state-of-the-art algorithms that solve the multi armed bandit problem. These algorithms were proposed by [16]. Unlike ϵ-greedy and softmax methods, these algorithms are based on sophisticated mathematical ideas and come with strong theoretical performance guarantees. We use 2 versions of the UCB family of algorithms: the UCB1 and the UCB1-tuned.

1) UCB1: The simpler algorithm, UCB1, maintains a count of the number of times scheme $n_i(t)$ scheme i was chosen. Initially, each scheme is played once. Subsequently at round t, the algorithm chooses scheme i as per:

$$j(t) = \arg\max_i \left(\hat{\mu}_i + \sqrt{\frac{2 \ln t}{n_i(t)}} \right) \tag{22}$$

2) UCB1-Tuned: UCB1-Tuned is an alternate version of the algorithm proposed by the authors in [16]. It can have better performance in practice, but it comes without theoretical guarantees. The main feature of the UCB1-Tuned is that it accounts for the variance of each scheme in addition to the expected reward. The schemes are chosen as per:

$$j(t) = \arg\max_i \left(\hat{\mu}_i + \sqrt{\frac{2 \ln t}{n_i(t)} \min(\frac{1}{4}, V_i(n_i))} \right) \tag{23}$$

where

$$V_i(t) = \hat{\sigma}_i^2(t) + \sqrt{\frac{2 \ln t}{n_i(t)}} \tag{24}$$

$\hat{\sigma}_i^2(t)$ is computed from the empirical sum of squares of $\hat{\mu}_i$.

VII. Results

A. Test setup

We compare the performance by varying the bandit selection algorithm used and plotting the average data rate. The x-axis of each plot represents time in terms of the number of packets transmitted. The y-axis represents the average data rate at that time instant. All data rates reported are user data rates, i.e. the total number of user data bits received divided by total time taken.

The handshake protocol necessary to exchange control information imposes a time overhead of the order of a 2-way propagation delay for every control packet exchange. Optimizing the protocol overhead is essential for obtaining high data rates. In particular, an underwater channel presents many opportunities for such optimization since one can exploit the large propagation delay time. See [27] and [28] for examples of such optimizations. However protocol optimization is not the focus our focus here. Hence, we ignore the protocol overhead in all results presented here.

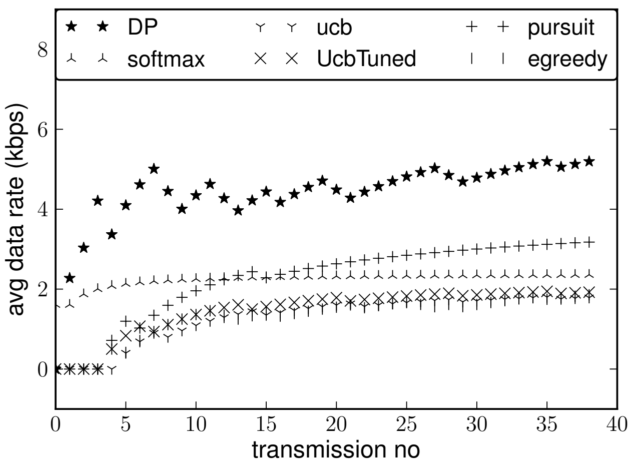

Fig. 6. Average data rate in simulation

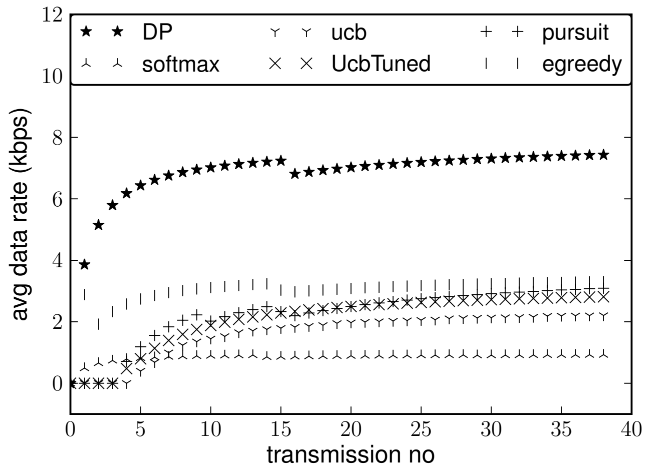

Fig. 7. Average data rate in tank

B. Simulation results

The majority of the tests over the course of the link tuner development were carried out on a simulator. The simulator tries to simulate an water tank environment. The simulator is a random forest regressor trained with data consisting of various schemes and the resulting BER as observed in a water tank. In order to train the regressor, we transmitted 3,546 schemes in the tank and recorded the uncoded BER. The regressor learns a function mapping from schemes to BER, and provides us with a BER estimate when presented with a scheme. In order to estimate the accuracy of the regressor, we performed a 5 - fold cross validation test. The data was randomly split between training (80 %) and test (20 %) sets. The regressor was trained on the training set, and its coefficient of determination was measured on the test set. The process was repeated 5 times, and the mean of the coefficient of determination was found to be 0.98, with a standard deviation of 0.45%.

Fig. 6 plots the average data rate attained in simulation. From the figure, we can see that the dynamic programming method attains the highest average data rate, followed by pursuit and softmax. UCB, UCB-tuned and ϵ-greedy have similar performance characteristics.

C. Watertank tests

Fig. 7 shows the performance of the tuner in a water tank. The tank measures $2{\times}2{\times}2$ meters. The modems were placed at diagonally opposite ends at depths of 0.5 meters. We can see that the dynamic programming algorithm has the best average data rate performance. The average data rate of the dynamic programming method exceeds the rest by 4 kbps. ϵ - greedy, pursuit, UCB-tuned, and UCB methods have average data rates between 2 to 3 kpbs. The lowest average data rate is recorded by the softmax method. As a benchmark, we also include the data rate of the default scheme (without tuning). The data rate yielded by the default scheme is 0.345 kbps. All methods yield data rates that are at least twice the default.

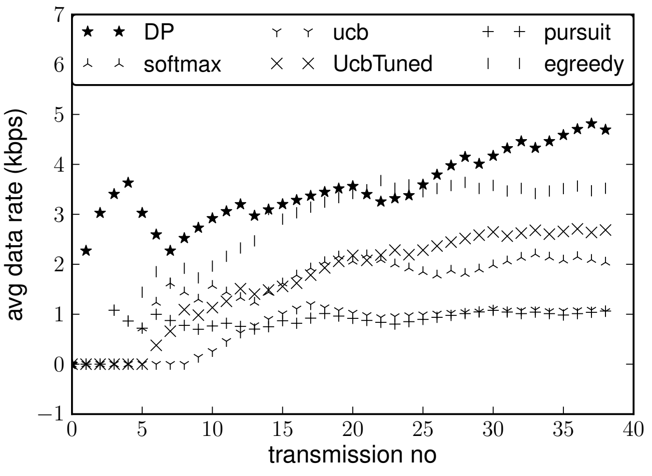

Fig. 8. Average data rate at RSYC

D. Republic of Singapore Yatch Club (RSYC) field trials

The RSYC is a marina which provides docking facilities for a number of yachts and boats. The presence of nearby boating activities makes the channel a very noise one to operate in. The water depth at RSYC is very shallow (6 to 7 meters on average). Both modems were deployed at 1.5 meter depth. These factors make RSYC a very challenging environment to operate in.

Fig. 8 shows the average data rates recorded at RSYC. We can see that the dynamic programming method has the highest average data rates for most of the test duration. This is followed by the $\epsilon-$ greedy, UCB-tuned and softmax methods. The pursuit and UCB methods have nearly identical as well as the lowest average data rates. We can benchmark our results by comparing against the data rate offered by the default scheme (without any tuning). The default scheme offers a data rate of 0.345 kbps. The average data rate of the tuner using the dynamic programming method is 4.7 kbps, representing a 13-

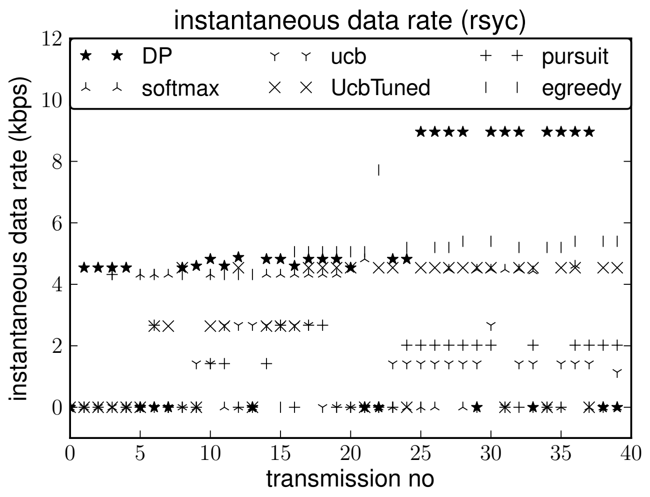

Fig. 9. Instantaneous data rates observed at RSYC

fold increase. The lowest average data rate attained by the pursuit method is 1.06 kbps, which is 3 times the default data rate. Fig. 9 shows the instantaneous data rate for the tests at RSYC.

VIII. Conclusion

Algorithms to tune a point to point underwater communication link were presented. A data driven approach to link tuning resulted in algorithms that need only bit error rate information as input, and are independent of the underlying modem implementation. An inherent exploration vs exploitation dilemma was identified and addressed. Results from experiments conducted in a simulation setup, a water tank, and field trials show significant improvements in average data rates attained.

IX. Acknowledgement

This work was partially funded through the Singapore Ministry of Education Academic Research Fund Tier 1 grant R-263-000-521-133.

References

[1] M. Chitre, I. Topor, and T. Koay, “The UNET-2 modem - an extensible tool for underwater networking research,” in *Proceedings of IEEE OCEANS’12 Yeosu*, 2012.

[2] J. Preisig, “Acoustic propagation considerations for underwater acoustic communications network development,” in *ACM SIGMOBILE Mobile Computing and Communications Review*, 2007.

[3] M. Porter and M. Siderius, “Acoustic propagation in very shallow water,” in *Waterside Security Conference (WSS), 2010 International*, 2010.

[4] P. Barley, H. Bucker, J. Rice, M. Green, and J. Woxstroem, “Acoustic communication channel modeling for the baltic sea,” in *OCEANS ’99 MTS/IEEE. Riding the Crest into the 21st Century*, 1999.

[5] S. Gao, L. Qian, D. Vaman, and Q. Qu, “Energy efficient adaptive modulation in wireless cognitive radio sensor networks,” *IEEE International Conference on Communications*, 2007.

[6] P. Tan, Y. Wu, and S. Sun, “Link adaptation based on adaptive modulation and coding for multiple-antenna OFDM system,” *IEEE Journal on selected areas in communications*, 2008.

[7] F.Peng, J.Zhang, and W. Ryan, “Adaptive modulation and coding for IEEE 802.11n,” in *Wireless Communications and Networking Conference*, 2007.

[8] C. Pandana, Y. Sun, and K. J. R. Liu, “Channel-aware priority transmission scheme using joint channel estimation and data loading for OFDM systems,” *IEEE Transactions on Signal Processing*, 2005.

[9] Y. Jing and X. Yu, “ML-based channel estimations for non-regenerative relay networks with multiple transmit and receive antennas,” *IEEE Journal on selected areas in communications*, 2012.

[10] K. Pelekanakis and M. Chitre, “Natural gradient-based adaptive algorithms for sparse underwater acoustic channel identification,” in *Proceedings of Underwater Acoustic Measurements: Technologies and Results*, 2011.

[11] C.Tang and V.Stolpman, “An adaptive learning approach to adaptive OFDM,” *Wireless Communications and Networking Conference*, 2004.

[12] T.Hastie, R.Tibshirani, and J.Friedman, *The elements of statistical learning*. Springer series in statistics, 2009.

[13] R. S. Sutton and A. G. Barto, *Reinfocement Learning: An Introduction*. MIT Press, 1998.

[14] D. A. Berry and B. Fristedt, “Bandit problems: Sequential allocation of experiments,” *Monographs on Statistics and Applied Probability*, 1960.

[15] J. Gittins, *Multi-armed bandit allocation indices*. Wiley-Interscience Series in Systems and Optimization, 1989.

[16] P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-time analysis of the multiarmed bandit problem,” *Machine Learning*, 2002.

[17] V.Kuleshov and D.Precup, “Algorithms for the mulit-armed bandit problem,” *Journal of machine learning research*, 2000.

[18] H. I. Volos, “Cognitive radio engine design for link adaptation,” Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2010.

[19] H. I. Volos and R. M. Buehrer, “On balancing exploration vs. exploitation in a cognitive engine for multi-antenna systems,” in *Proceedings of the IEEE Global Telecommunications Conference*, 2009.

[20] ——, “Cognitive engine design for link adaptation: An application to multi-antenna systems,” *IEEE Transactions on Wireless Communications*, 2010.

[21] S. Shankar, M. Chitre, and M. Jayasuriya, “Data driven algorithms to tune physical layer parameters of an underwater communication link,” in *Proceedings of IEEE Oceans Sydney*, 2010.

[22] D. M. Jayasuriya, “Adapting underwater physical link parameters using data driven algorithms,” Master’s thesis, National University of Singapore, 2010.

[23] R. Kalman, “A new approach to linear filtering and prediction problems,” *Transactions of the ASME-Journal of Basic Engineering*, 1960.

[24] G. Welch and G. Bishop, *An Introduction to the Kalman Filter*. TR 95-041, Department of Computer Science University of North Carolina at Chapel Hill, 2006.

[25] R. Bellman, *Dynamic Programming*. Princeton University Press, Princeton, NJ. Republished 2003: Dover, ISBN 0-486-42809-5., 2003.

[26] J. Vermorel and M. Mohri, “Multi-armed bandit algorithms and empirical evaluation,” in *Proceedings of ECML*, 2005.

[27] M.Chitre and W. Soh, “Reliable point-to-point underwater acoustic data transfer: To juggle or not to juggle?” *IEEE Journal of Oceanic Engineering (under review)*, 2013.

[28] R. Bhatnagar, *Implementing and testing of ARQ protocols for underwater networks*. National University of Singapore EE5003 Project Report, 2003.