ADR-X: ANN-Assisted Wireless Link Rate Adaptation for Compute-Constrained Embedded Gaming Devices

Hao Yin* *University of Washington*
Murali Ramanujam* *Princeton University*
Joe Schaefer *Microsoft*
Stan Adermann *Microsoft*
Srihari Narlanka *Microsoft*
Perry Lea *Microsoft*
Ravi Netravali *Princeton University*
Krishna Chintalapudi *Microsoft Research*

Abstract

The wireless channel between gaming console and accessories e.g. controllers and headsets, experiences extremely rapid variations due to abrupt head and hand movements amidst an exciting game. In the absence of prior studies on wireless packet losses for console gaming, through extensive evaluations and user studies, we find that state-of-the-art rate adaptation schemes, unable to keep up with these rapid changes, experience packet loss rates of 2-10% while loss rates that are 10× lower (0.1-0.5%) are required to ensure a high quality gaming experience. We present ADR-X, an ANN-based contextual multi-armed bandit rate adaptation technique that continuously predicts and tracks the channel and picks appropriate data rates. A key challenge for ADR-X is that it must run on power and compute constrained embedded devices under realtime constraints. ADR-X addresses this challenge by meticulously crafting an ANN that leverages existing communication theory results to incorporate domain knowledge. This allows ADR-X to achieve 10× lower packet losses than existing schemes while also running 100× faster than state-of-the-art reinforcement learning schemes, making it suitable for deployment on embedded gaming devices.

1 Introduction

Gaming is a $200 billion industry experiencing a rapid upward trajectory in growth. A typical gaming setup comprises a gaming console/PC located within 10 feet of the gamer. During gameplay, various gaming accessories such as controllers with joysticks and buttons [42, 65], headphones or specialized headsets for audio, VR, or AR [43, 66] connect to the console either via cables or wireless (Figure 1). The console continuously receives player inputs, such as button interactions and joystick maneuvers, and adjusts the game state in real-time. Subsequently, it generates and dispatches audio, video, and tactile feedback (like haptic responses) to the respective connected accessories. The accessories are typically power and compute constrained embedded devices e.g. Xbox controllers use 250-600MHz ARM based processors [1].

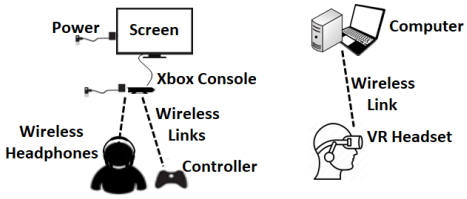

Figure 1: A typical gaming setup.

Despite wired headsets/headphones impeding free movement, ardent gamers today, prefer wired over wireless connectivity as Wi-Fi[1] packet losses deteriorate game audio/video quality. Prior user studies have determined that packet loss rates $< 0.5\%$ are necessary for a high quality gaming audio/video experience [58, 70]. However, these studies were conducted in the context of internet game streaming assuming independent (Bernoulli) packet losses over wired connections. To the best of our knowledge, no prior study examines the nature and effect of packet losses in a console gaming session over wireless links between the accessories and the console.

Amidst an exciting game (e.g. car racing or first person shooter games), the constant jerky hand (controller) and head (headset) movements of the gamer, rapidly affect the wireless channel resulting in up to $15dB$ signal strength variations within 50-100ms (Sec 3.2) – $5-10\times$ faster than usual Wi-Fi usage scenarios. We find that these rapid changes result in increased packet losses as the wireless Adaptive Data Rate (ADR) schemes are too slow to react. We also find that packet losses occur in succession, pairs and triplets, $\approx 50\%$ of the time (rather than independently as previous user studies assumed). *We conduct the first user study to quantify the effect of successive packet losses on gaming audio/video experience and find that (Sec. 3.3) consecutive losses are in fact more perceptible than random isolated losses; e.g. with 2-3 consecutive losses, gamers negatively perceive loss rates as low as*

* These authors contributed equally to this work.

[1] Although some existing gaming platforms adopt Bluetooth Low Energy (BLE) for connectivity, low throughput and high congestion in 2.4 GHz results in large latency, poor scalability.

0.1% for VR, a harsher requirement than that suggested by prior studies. **We have obtained an IRB for this study.**

While there exists a vast literature of wireless ADR schemes (Sec. 2.2), they have neither been tested nor designed for these severe gaming conditions. It is therefore no surprise that in our extensive evaluations (Table 1) of the state-of-the-art ADR techniques, as well as from traces collected from off-the-shelf gaming devices, we find $1-10\%$ packet loss during an active gaming session, falling well short of the desirable packet loss target.

Our key contribution is a novel wireless ADR technique, ADR-X, that achieves $10\times$ *lower packet losses (0.1-0.3%) than state-of-the-art during gaming conditions and an order of magnitude lesser consecutive packet losses by "closely" following and predicting the wireless channel dynamics.* ADR-X takes a hybrid approach – it leverages the well established analytical results from communication theory alongside an Artificial Neural Network (ANN) based contextual multi-armed bandit. It employs online learning to implicitly model, predict and adapt continuously to host radio hardware, background interference conditions and gamers' physical movement patterns that are hard/impossible to model analytically. As we demonstrate in our evaluations (Sec. 5.2), reinforcement learning approaches that do not leverage domain knowledge (e.g. PPO [60]) run $100\times$ slower than ADR-X and are unable to meet the real-time and low compute constraints imposed by the embedded gaming accessories.

Designing an ADR scheme that can predictively determine the ideal data rate under the fast changing gaming channel conditions poses two key challenges:

Wireless gaming channels are hard to predict analytically. Communication theory provides analytical functions that can accurately predict packet error rates over a wired medium [50]. Wireless channels however are far less amenable to modelling and prediction due to frequency selective fading that changes rapidly as the gamer changes hand and head orientations during gameplay and unpredictable background interference. As we discuss in Section 2.2 prior attempts [18] to analytically predict packet loss are extremely sensitive to small calibration and measurement errors, background interference, wireless channel asymmetry due to differences between transmitter and receiver radios, variations due to radio circuit components such as Automatic Gain Control (AGC), etc. ADR-X leverages the capability of ANNs to model and adapt to the intractable effects of the wireless channel and gamer's movements in a data-driven manner.

Computation, power and real-time Constraints. (Sec. 4.2) Gaming accessories e.g. controllers, headphones and headsets, are low power embedded devices with limited computation capabilities. For example, Xbox controllers uses 250MHz ARM processors [42] and employ sleep duty-cycling roughly 1 out of 8ms to save power. A naive strawman approach might be to employ ANN based online reinforcement learning to predict the appropriate data rate. However, as our evaluations show (Sec. 5.2), this approach proves to be extremely computationally demanding and unsuitable for embedded gaming devices. The key insight in the design of ADR-X is to rely on an ANN to model only the intractable effects while leveraging the predictive power of analytical models used in wired communication channels. This relieves the ANN from the burden of re-learning well-known communication theoretical results, making it more accurate and computationally economical.

ADR-X overview. ADR-X employs a contextual multi-armed bandit architecture (Sec. 4.1) that comprises three stages. The first stage is an ANN that transforms the wireless channel measurements (e.g. Signal to Noise Ratio (SNR), Channel State Information (CSI)) into an equivalent wired channel conducted SNR – CSNR, by implicitly and continuously learning to model the latent effects such as background interference, calibration errors, wireless channel asymmetry, hardware/firmware specifics and gamer movements. The second stage then leverages communication theory results to predict packet success rates for each of the possible data rates. In practice, to facilitate efficient gradient descent-based learning, we approximate the complex analytical communication theory models in the second stage using sigmoid functions (used commonly in training ANNs) with fixed pre-computed parameters. Finally, the third stage uses an $\varepsilon-$greedy sampling approach commonly used in multi-armed bandits to pick a suitable data rate. Further, as described in Section 4.2, ADR-X employs Wi-Fi domain specific feature engineering for the ANN to allow for a small ANN architecture.

Dealing with survivorship bias. When packets are lost, so are the associated channel measurements used to train the ANN. This leads to survivorship bias [63] i.e. the ANN learns only from successful examples leading to poor performance. To solve these problems ADR-X uses a specially crafted re-transmission strategy (Sec. 4.4) with stepped reductions in data rate for each subsequent re-transmission that serves as "training wheels" by providing second (or third) chances. When an ACK is received after one (or more) retries, ADR-X obtains negative examples corresponding to prior losses and channel state. As ADR-X learns, its reliance on these "training wheels" diminishes. Further, ADR-X uses imputation techniques [14] such as interpolation when packets are lost despite the re-transmissions.

Quick start using pretrained/federated models. Starting from a random initialization, when the player plays for the first time, ADR-X takes about 50s of gameplay to learn and converge (Sec. 5.3). We find that techniques such as (i) pre-training the ANN on IEEE channel simulations, and (ii) using federated model weights from other gamers or prior sessions reduce this to about 20s. Further, this time reduces to about 10s after a few gaming sessions. The stepped re-transmission strategy (described earlier) helps ADR-X in these initial few seconds to overcome high packet losses.

Summary of contributions.

- We provide the first user study and insights on the nature of packet losses for console-accessory wireless links and their effect on the gaming audio/video experience (we have obtained an IRB for the study). We show that existing wireless ADR schemes are unable to adapt "fast enough" in response to the high dynamism of gaming wireless channels resulting in successive losses rather than isolated incidents. To the best of our knowledge, our user study is the first to highlight that the traditionally accepted target loss rate of 0.5% for individual losses should be supplemented by 0.1% for successive losses.
- We propose ADR-X, an ANN assisted contextual multi-armed bandit based novel adaptive data rate scheme that is suited for power and compute constrained embedded gaming accessories. ADR-X predictively adjusts data rates based on the history of channel wireless measurements and packet losses.
- We evaluate ADR-X extensively across 20 diverse games[*], including multiple genres such as First Person Shooter (FPS), racing, and action. In real-world experiments with practical hardware, under extensive comparisons against a bevy of state-of-the-art ADR baselines including several ML based approaches, ADR-X is the only scheme to achieve $0.1-0.3\%$ packet loss on hardware matching Xbox accessory clock speeds of 250-600MHz.

2 Background and Related Work

Signal to Noise Ratio (SNR) η, determines the data rate at which bits can be successfully sent over a communication channel. A data rate too high causes bits to be lost, and a data rate too low is wasteful in terms of time and energy. Since wireless channel conditions vary significantly over time; Adaptive Data Rate (ADR) techniques aim to adapt data rate to these changes by choosing the most appropriate data rate. The Modulation Coding Scheme (MCS) index, a number between 0 to 9, in Wi-Fi determines the data rate of transmission – the higher the MCS index, the higher the data rate. For each MCS index m, communication theory allows the bit error rate to be calculated analytically using a function $ber_m(\eta)$ [16].

2.1 Channel Measurements

In OFDM [49] modulation used by Wi-Fi, a 20MHz channel is split into 52 sub-carriers (sub-channels) in 802.11n and 242 sub-carriers in 802.11ax. Bits in a packet are spread out and transmitted over these sub-carriers.

CSI. A radio wave transmitted over the i^{th} sub-carrier undergoes changes in amplitude and phase represented by a complex number c_i. Channel State Information (CSI) collects all these values into a vector $\mathbf{c} =< c_1, \cdots, c_C >$. While all Wi-Fi radios have to necessarily measure CSI for each received packet to decode, not all radio hardware/firmware provides access to CSI information through an API.

RSS. Almost all radios provide Received Signal Strength (RSS) measured in `dBm`, for each received packet – this measures the power level (strength) of a received signal.

SNR. Some radio APIs provide SNR, η (in `dB`) for each packet. SNR and RSS in `dB` are related to each other as [18]

$$\mathbf{SNR} = \mathbf{RSS} - \mathbf{NF} - \mathbf{AGC} \tag{1}$$

In Eqn. (1) NF is the noise floor of the radio and AGC is Automatic Gain Control of the radio which is a dynamic gain introduced by the radio circuit. While some radio APIs provide Noise Floor (in `dBm`) of the radio [22, 52], they do not provide AGC. In practice computing SNR using RSS and NF without accounting for AGC can lead to several `dB` of error.

SNR per subcarrier. As prior work [18] demonstrates, SNR is an extremely poor predictor for packet loss due to frequency selective fading, as each sub-carrier experiences a different SNR and hence experiences a different bit-error rate. The success or failure of a packet depends on the aggregate success of all the bits transmitted across all the sub-carriers. The SNR of the k^{th} sub-carrier, η_k can be computed by scaling η by the k^{th} component of unit vector of c,

$$\begin{aligned} \zeta_k &= \frac{C}{\sum_i \|c_i\|^2} \|c_k\| \\ \eta_k &= \eta - 10\log_{10}\zeta_k \end{aligned} \tag{2}$$

Effective SNR (ESNR). In channels undergoing frequency selective fading, to tackle the ineffectiveness of SNR in predicting packet rate, [18] introduces ESNR (η_{esnr}). It summarizes the per-subcarrier values $< \eta_1, \cdots, \eta_C >$ into a single value to represent an equivalent SNR value corresponding to a flat-fading channel (a channel without frequency selective fading) by treating the wireless channel to be composed of several narrow wired channels, one for each sub-carrier. Thus, for each MCS index m, [18] suggests computing η_{esnr} analytically by calculating the average bit rate across all the sub-carriers,

$$\eta_{esnr} = ber_d^{-1}\left(\sum_{i=1}^{i=C} ber_d(\eta_i)\right) \tag{3}$$

ESNR Computation is sensitive and error prone. Calibration errors and measurement errors of a few `dB` in CSI, RSS, noise floor, or SNR are common in all radios. Further, ber_m functions are exponential (based on the Q function) with very sharp transitions from 0 to 1 within 2-3 `dB`. This makes ESNR computation extremely sensitive to even small errors – 2-3 `dB` error can result over 10`dB` error in effective SNR. Consequently, subsequent work [13] uses an exponentially weighted mean of η_i, which is estimated during an initial calibration phase, rather than relying on Eqn. (3).

[*] Full list of games with media samples exhibiting the impact of packet losses is at https://muralisr.github.io/ADRX/.

2.2 Existing ADR Techniques

In this paper, we broadly classify ADR techniques into *reactive* and *proactive* ADR techniques. The former relies on recent packet loss and re-transmission statistics (e.g. using running average estimates). The latter makes use of channel measurements such as RSS, SNR, CSI etc. to make a timely choice based on the current state of the channel. Further, several recent ADR techniques, both reactive and proactive, employ ML to cope with vagaries of the wireless channel.

Reactive techniques. ARF [27], AARF [35], and CARA [31] gradually increase or decrease the rate based on the success or failure of consecutive transmission results. Several rate control algorithms are designed to optimize specific metrics instead of losses only *e.g.* average transmission time [5, 71], frame loss ratio [36, 48, 73, 74], bit error rate [64, 69] and throughput [9, 17, 47, 72, 75].

Reactive techniques Using ML. NeuRA [30] and MLRA [39] attempt to reduce sampling overhead by using a neural network model to predict the throughput of unsampled data rate. Thompson Sampling [34] uses a multi-armed bandit approach to improve the sampling of different data rates. As discussed in Section 3.2, reactive techniques in general are too slow and are unable to keep pace with the channel dynamism during gaming, leading to a poor gaming experience.

Proactive or channel measurement-based techniques. [33] uses RSS while [12, 21, 26, 38, 54, 56, 61, 68] use SNR measurements from received packets to dynamically adjust transmission rates. As discussed in Section 2.1 the performance of these schemes is limited by the fact that RSS and SNR do not take into account the effects of frequency selective fading. While ESNR-based techniques [18] account for frequency selective fading, as discussed in Section 2.1 their performance is limited due to the sensitivity of ESNR computation to hardware calibration errors and the requirement of offline calibration for each device.

Proactive schemes using ML. Recently, researchers have demonstrated the potential use of ML in proactive schemes through Simulation studies [28, 29, 39, 53]. To the best of our knowledge, EDRA [11] is the only existing implemented proactive ML-based ADR scheme. EDRA uses reinforcement learning, aiming to maximize throughput through joint rate and bandwidth adaptation by using Deep Q-Learning [19] with SNR, loss rates, and service times as inputs. However, EDRA imposes severe computational requirements – even a single inference takes 1.3-3.7ms on high-performance CPUs like i7-8700 and i5-6200U.

3 Packet Losses in Gaming

In this section, we analyze wireless packet losses during active gaming sessions. We show how off-the-shelf ADR schemes are unable to keep up with the highly dynamic gaming wireless channel and cause multiple consecutive packet losses. We then describe our user studies that show users are more sensitive to multiple consecutive losses than isolated ones.

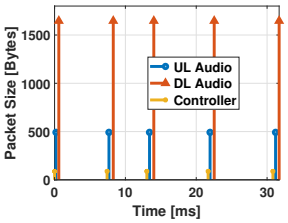

Figure 2: Xbox traffic.

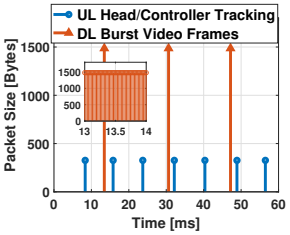

Figure 3: VR traffic.

3.1 Gaming Traffic Patterns

To understand packet flow in gaming traffic, we collect packet traces for Xbox [41] and Oculus [40] Quest 2 using a sniffer during multiple gaming sessions – we highlight the results from two games (out of the 20 games in our corpus) CrossFireX (Xbox) and Robo-Recall (Oculus Quest 2), but we note that trends persist across games. As seen in Figure 2, the Xbox console transmits one PCM game audio packet (1646 bytes), receives one chat audio packet (492 bytes) and one game input packet (88 bytes) every 8ms. In VR traffic (Figure 3), a burst of roughly 50 packets (total of about 500Kb) comprising game video and audio are transmitted once every 16.6ms (corresponding to 60Hz video frame refresh rate). A headset tracking packet (326 bytes) is transmitted every 8ms.

3.2 Packet Losses During Gaming

In this sub-section, our goal is to gain insights into how wireless packet losses occur during an active gaming session.

Figure 4: The setup for console gaming experiments.

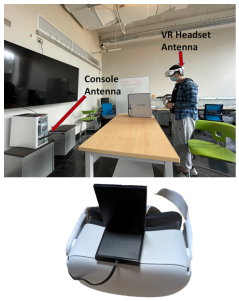

Figure 5: The setup for VR gaming experiments.

Measurement methodology. In order to emulate Xbox and Oculus Quest 2 radio firmware for controlled experiments, we collect and replay game traces between two PCs. We affix antennas on the console/desktop and the controller (since players connect to the headset via an audio jack from the controller) and VR headset (Figure 4, 5). We ask the gamers to play a game using the controller/headset with the affixed antenna. This allows the antenna to experience the same head/hand motions as the headset/controller. When actual gaming traffic is exchanged between the controller/headset and the console/desktop during a gaming session, we transmit gaming traffic traces in an interference-free DFS channel [45] between

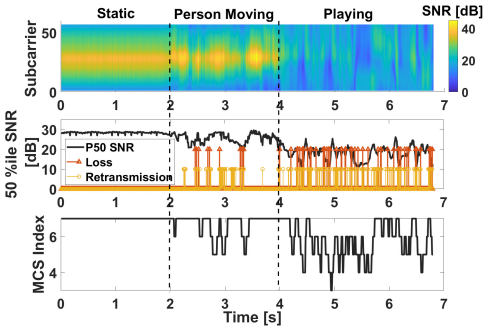

Figure 6: Channel, 50%ile SNR, re-transmissions, Losses and MCS for Xbox Traffic.

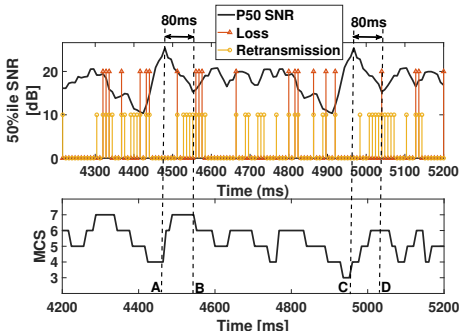

Figure 7: Rate adaptation is unable to keep up with rapid channel changes causing a cluster of losses.

the two PCs. We use PicoScenes toolbox [24, 25] to capture the transmitted packets and measure their CSI and SNR. We use these measurements to compute individual SNRs for each of the 52 Wi-Fi sub-carriers as described in Section 4.2. We consider three scenarios.

- *static* – the controller/headset is static, on a table about 5 feet from the transmitter in an empty room
- *people movement* – the controller/headset is static on a table 5 feet from the transmitter but with a person walking around in the room.
- *game play* – a gamer actively plays using the controller/ headset 5 feet from the transmitter (Figure 4, 5).

Observations. Figure 6 depicts the SNR heatmap for each of the 52 Wi-Fi sub-carriers spanning a 20MHz Wi-Fi channel as a function of time for each of the three scenarios. Figure 6 also depicts the instantaneous 50%ile SNR across all the sub-carriers. This is based on the intuition that a packet loss in Wi-Fi will occur when a "significant" fraction of sub-carriers experience fading (low SNR) so that error correction is unable to recover the correct bits. We also plot packet re-transmission events and loss events (when the re-transmission fails) and the MCS index (data rate) chosen by the native ADR scheme in an off-the-shelf device.

In the static scenario, the wireless channel is excellent with all the sub-carriers having an SNR of 30dB or higher with no losses or re-transmissions. In the people movement scenario, the channel sees variations with time due to changes in multi-path reflections, and occasional packet losses. When the gamer holds the controller or wears the headset and starts playing the game, the channel changes rapidly, leading to a large number of re-transmissions and packet losses. The ADR scheme shows large frequent variations in MCS Index.

Data rate adaptation is unable to keep up. Figure 7 shows a zoomed 1s section of the wireless channel for the playing scenario between 4.2 to 5.2 seconds. The 50%ile SNR across subcarriers shows a variation of over 15dB with a rapid decline of up to 10dB within 80ms in sections AB and CD. *The interesting observation in Figure 7 is that while the channel is degenerating, as seen by the decreasing trend in the 50%ile SNR in AB and CD, the ADR scheme actually increases data rates (MCS Index) and then starts reducing it when the channel SNR is improving – the exact opposite of the desired behavior.* This is because of the native ADR scheme, which is *reactive*, i.e. relies on recent packet re-transmission and loss statistics (e.g. running average) to adapt data rate. Before A, the channel 50%ile SNR improves by almost 15dB within a matter of 70ms. Based on the running average, after A, as the channel deteriorates, the data rate is increased causing a cluster of packet re-transmissions and losses. After B, even though the channel has an improving trend, data is decreased due to the history of failures.

Consecutive packet losses. As seen from Figure 7, due to ADR's reaction lag, packet losses occur in clusters and consecutive packets are lost when channel conditions change rapidly. Figure 8 shows the distribution of the number of consecutive losses for different distances between the controller and console. As seen from the figure, about 55-65% (100%-fraction of single packet losses) of all the packet losses occur with two or more packets lost consecutively. Our experiments with the VR headset (elided for space constraints) also show a similar cluster of losses.

3.3 Effect of Consecutive Packet Losses

Prior studies [7, 10, 57, 58, 70] in the context of internet game streaming have shown that Bernoulli (independently occurring) packet loss rates beyond 0.5% deteriorates audio/video quality for online gaming. However, to the best of our knowledge, there has not been a study quantifying the effect of **consecutive** packet losses on audio/video quality in a wireless console gaming setup. Thus, we conduct a study to measure the effect of consecutive packet losses on gaming audio and video.

Test data. For our study, we chose 30 Xbox game audio clips each 15s long, and 60 VR game video clips each 5s long as our original data set. These clips were drawn from the games in our corpus comprising 15 popular Xbox games and 5 popular VR games including racing, First Person Shooter (FPS), and other action games(full list of games with media samples exhibiting the impact of packet losses is at https://muralisr.github.io/ADRX/). Audio data is streamed as PCM audio packets (similar to Xbox) with

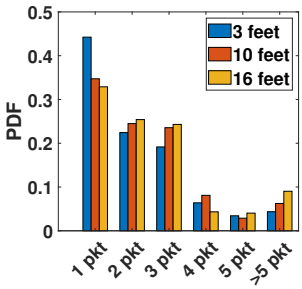

Figure 8: Consecutive packet losses due to slow rate adaptation.

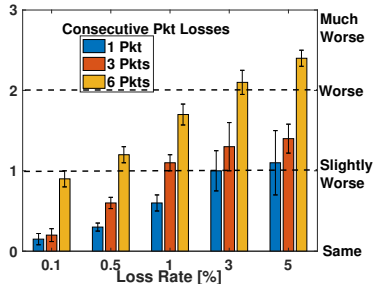

Figure 9: Effect of packet losses on game audio.

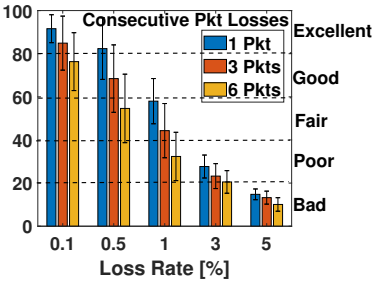

Figure 10: Effect of packet losses on game video.

8ms worth of data in each packet. Video data stream is encoded using H.264 video compression. Losses are injected for each stream randomly, parameterized by different loss rates $\rho = \{0.1\%, 0.5\%, 1\%, 5\%, 10\%\}$ and consecutive loss length $b = \{1, 3, 6\}$ packets. These parameter choices are informed by the loss patterns observed during our experimental evaluations (Sec. 5) of existing ADR schemes. This gives us 15 different combinations of $< \rho, b >$. Once a packet is randomly dropped with probability ρ, $b-1$ following packets are also dropped to capture consecutive losses during rate adaptation.

Measuring effect on game audio We conduct a user study adhering to the Comparative Mean Opinion Score (CMOS) test methodology as dictated by the ITU-T P.808 standard [44]. CMOS, as a subjective measure based on human perception, was chosen for its ability to accurately assess user-experienced audio quality degradation due to packet losses, particularly relevant for gamers. In contrast, PESQ [55], an objective algorithmic measure, lacks this direct user experience perspective, making CMOS more suitable for evaluating relative audio quality in our context. IRB approval from the author's organization was obtained prior to this study. Ten unique participants were recruited to listen and compare two audio clips – a lossy and an original clip, without knowing which clip was the original. On a scale from -3 (much worse) to +3 (much better), participants scored audio quality. During the study, care is taken to sanitize the results for faulty data points by eliminating incoherent outputs, and a calibration step is used to ensure proper audio setup on the participants' computers before the experiment (as per ITU-T P.808). Each combination of $< \rho, b >$ received 300 votes in all (30 clips $\times$10 people). As seen from Figure 9, our study reveals that listeners are more sensitive to consecutive packet losses – deterioration becomes perceptible at even 0.5% loss if $b \geq 6$.

Measuring effect on game video To quantify the effect of packet losses on VR video, we use the Video Multimethod Assessment Fusion (VMAF) [46]. VMAF is a popular full-reference objective video quality assessment model developed by Netflix that uses human-vision modeling. VMAF predicts a quality score that ranges from 0 to 100 for each video. Figure 10 shows that $\rho \leq 0.5$ and $b \leq 3$ are required to ensure the highest perception quality.

3.4 Conclusions

We summarize the observations in this section as,

- During gaming, the wireless channel varies by 10-15dB within a matter of 50-100ms.
- Rate adaptation is unable to keep up with these rapid changes resulting in consecutive packet losses.
- User studies indicate more than 2 consecutive packet losses causes "significant" visual artifacts even at the widely accepted standard of 0.5% packet loss rate. During gaming, consecutive packet losses become imperceptible only at the stricter threshold of 0.1%.

4 ADR-X

ADR-X continuously tracks the wireless channel and adapts data rates predictively using the recent history of channel measurement time series to avoid incurring losses. The fact that gaming traffic is periodic 8ms-16ms (Sec. 3.1) facilitates this approach since wireless channel measurements (e.g. CSI and SNR) can be measured each time a packet is received. ADR-X combines the strengths of ANNs to model the unmeasurable or hard-to-measure effects while leveraging the analytical simplicity of predicting packet loss over wired channels. This approach allows ADR-X to be both accurate and computationally efficient.

4.1 Overview of ADR-X

ADR-X takes a contextual multi-armed bandit approach – where a feature vector derived from the recent history of channel measurements serves as context (Sec. 4.2). ADR-X comprises three logical parts – i) *CSNR Mapper*, an ANN with learnable parameters that uses online gradient descent based learning [6] to transform the context into Conducted SNR (CSNR), ii) *PSR Calculator*, a layer pre-trained using standard communication theory results to predict the packet success rate given CSNR, and finally iii) *MCS Sampler*, an explore-exploit module using multi-armed bandit approaches to select the appropriate transmission rate to facilitate online training.

CSNR Mapper. CSNR in spirit is the same as ESNR (Sec. 3) – it maps the wireless channel measurements to a wired equivalent SNR. However, while ESNR is computed analytically,

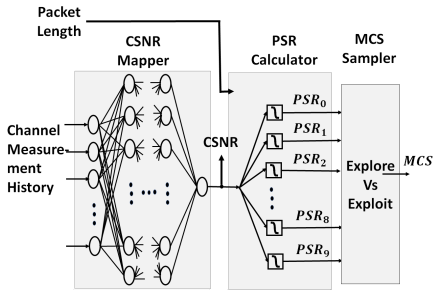

Figure 11: Architecture of ADR-X

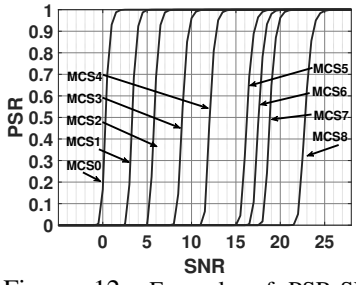

Figure 12: Example of PSR-SNR Curves for L=1536 over 802.11n.

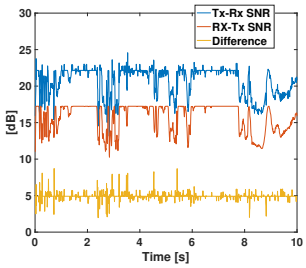

Figure 13: Channel asymmetry

treating each sub-carrier as an individual wire (Eqn. (3)), CSNR uses an ANN to learn this mapping on the fly in a data-driven manner. In this way, CSNR hopes to account for several practical hard to measure or un-measurable latent effects inherent in wireless communication – such as calibration and measurement errors, unknown AGC (Automatic Gain Control) or noise floor, background interference levels, wireless channel asymmetry, etc. as well as the nature of player's movements.

The CSNR mapper takes an input feature vector $\mathbf{f} =< f_1, f_2, .., f_n >$ extracted from the time series of channel measurements e.g. CSI, SNR, RSSI, etc. (as described in Section 4.2) and predicts the CSNR η_{csnr} in dB as,

$$\eta_{csnr} = F(\mathbf{f}, (\Theta)) \tag{4}$$

In Eqn. (4), Θ are learnable parameters – weights, biases etc.

PSR Calculator. Since CSNR represents the SNR for a flat fading channel. Communication theory provides analytical formulae $ber_m(\eta)$ to predict bit-error rate corresponding to MCS index m (data rate) and SNR η (Sec. 2). Packet Success Rate (PSR) may then be computed using $ber_m(\eta)$ and packet length L (Figure 12). These PSR functions however, tend to be complex and hard to compute analytically and typically fractional powers of $erfc(x)$ [2]. For enabling gradient descent based learning in ADR-X then, we would need to compute analytical expressions for the derivatives of these PSR functions, which are expensive to compute. Instead, ADR-X observes that the PSR-SNR curves qualitatively resemble a typical switching function [4] e.g. a Sigmoid function [8]. Thus, in ADR-X, the PSR Calculator approximates the PSR-SNR curves using the Sigmoid functions, a function commonly used in Neural Networks. The packet success rate is computed as,

$$PSR(m,L) = \frac{1}{1+e^{-(\alpha_{m,L}+\beta_{m,L}\eta_{csnr})}} \tag{5}$$

In Eqn. (5), $PSR(m,L)$ is the estimated PSR for the data rate corresponding to MCS index m, and the number of data bytes L. We first use Matlab IEEE flat fading model simulations to compute PSR for various values of SNR for different values of $< m, L >$. Then we estimate $\alpha_{m,L}$ and $\beta_{m,L}$ as a Least Mean Square (LMS) estimate to fit these values. Since there are too many possible values of L, in our implementation we compute three sets of PSR-SNR curves for broad L ranges – (0-64 bytes], (64-512 bytes] and (512-1536 bytes] by generating SNR, PSR values randomly sampling over the range of L. $\alpha_{m,L}$ and $\beta_{m,L}$ remain fixed during online training in ADR-X.

MCS Sampler. Once the PSR for each MCS Index is computed, we must choose the appropriate data rate. In ADR-X we define the optimal MCS value $MCS*$ as one that offers PSR higher than a certain threshold θ_{PSR},

$$m* = argmax_m PSR(m,L) > \theta_{PSR} \tag{6}$$

Since we aim to achieve $< 0.5\%$ PSR, we choose $\theta = 0.999$. Since the wireless channel changes with time, as described in Section 3.2, ADR-X must continuously explore to adapt and train correctly. Always sampling too conservatively will lead to ADR-X choosing lower MCS values resulting in increased power consumption and wastefully long transmission times. Thus, ADR-X must occasionally take risks and explore higher MCS rates. This problem is well known as the explore versus exploit problem in reinforcement learning literature and there exist several techniques [3, 20] to handle this tradeoff. In our implementation, we choose an ε-greedy approach as it works well in practice. $MCS*$ is chosen with the probability of ε (0.9 in our implementation) and a rate one step higher with a probability of $1-\varepsilon$.

4.2 Feature Engineering

Since ADR-X uses an ANN, it can potentially be generalized to use different kinds of channel measurements. Almost all radios provide RSS while some others provide CSI in addition. Thus, we generate two different flavors of features depending on the available channel measurements – ADR-X(CSI, RSS) and ADR-X(RSS). ADR-X(RSS) simply uses a time-series history of recently measured RSS value samples that are 8ms/16ms apart. ADR-X(CSI, RSS) uses the cumulative distribution function (CDF) of per-subcarrier SNR (η_i) described in Section 3. For computing SNR from RSS, we simply use NF as a constant -101db and let the ANN implicitly model the errors in the SNR.

For ADR-X(CSI, RSS), the time series history of per-subcarrier SNR values (Section 3) can lead to a large input to the ANN which can be either 52 (802.11n) or 242 (802.11ax) per time step. ADR-X however, must run on power and computation-constrained embedded devices such as 250-600MHz ARM processors (used in Xbox controllers) [1, 42].

To achieve this, we engineer the features to reduce the burden on the ANN itself in three steps.

1. Reducing frequency resolution. Instead of using all 52 or 242 values of per-subcarrier SNR, we compute the average SNR over spectrum bins each 2MHz wide (computing an average over the constituent subcarriers in each bin) to generate a vector $\mathbf{v} =< v_1, \cdots, v_9 >$ with nine values. This is based on the observation that magnitudes of CSI of sub-carriers that are "close" to each other in frequency are typically similar i.e. correlated.

2. Sorting SNR values. We sort the nine values of $\mathbf{v}$ in a decreasing order before presenting to the ANN - to create a CDF representation of the values - $\mathbf{v}^{sorted}$.

3. Including historical time series. Since the per-subcarrier SNRs show clear trends in 50-150ms time scales, we wish the ANN to implicitly extrapolate the current channel quality and compute PSR values. Thus, we provide the ANN with a time series history of $\mathbf{v}^{sorted}$ to allow it to make use of channel trends. In our implementation, we found that using channel information from 3 previous provides the most benefit. The feature vector at time t, $\mathbf{f}^t$ is computed as,

$$\mathbf{f}^t = \left[\mathbf{v}^{sorted}_{t-(H-1)\Delta}, \mathbf{v}^{sorted}_{t-(H-2)\Delta}, \cdots, \mathbf{v}^{sorted}_{t}\right] \tag{7}$$

Here, H is the length of the time series and the feature vector $\mathbf{f}$ comprises $9H$ numbers. Through experimentation, we chose $H = 3$ for our implementation. When ADR-X loses packets, it does not receive channel measurements. In this case, we use linear interpolation on the historical CSI values to fill in the gaps as imputation.

4.3 Online Training of ADR-X

ADR-X uses online training to allow it to adapt to changes in wireless channel conditions.

Training Data. Each time a packet is transmitted, the receipt of ACK or lack thereof indicates the success or failure of the transmission. Thus, after each transmission, we obtain the data $< L, m, r >$. Here L is the length of the packet, m is the MCS index used and r is a binary variable equal to 1 if the ACK was received or 0 if the packet was not received.

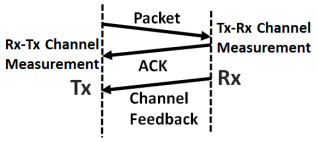

Figure 14: Channel measurements and feedback. ADR-X does not use feedbackbut relies on the reverse channel measurements.

Obtaining RSS, CSI and wireless channel asymmetry. CSI and SNR are obtained at the receiver (Rx) when a packet is received, however, ADR-X runs on the transmitter (Tx) (Figure 14). In principle, the receiver can explicitly provide a channel feedback message with CSI and SNR information (as employed in other proactive schemes). This approach constitutes an additional communication overhead and complexity of transmitting feedback. ADR-X, however, uses the channel measurements from the received packets (or ACKs) on the reverse path i.e. Rx-Tx instead of Tx-Rx. While in theory they are supposed to be the same, in practice, these measurements will be different (Figure 13), largely due to power differences but also due to AGC and hardware/firmware differences. As seen from Figure 13, there is a mean 5dB offset between the Tx-Rx and Rx-Tx channel SNRs of two devices along with a variation of up to $\pm$ 5dB. ADR-X implicitly learns to model the channel asymmetry in-situ by using online training methods to learn the PSR from real transmissions. This allows it to adapt to deployment scenarios with varying characteristics and generalize to different devices.

Loss function and weight updates in ADR-X. Suppose that the k^{th} packet was transmitted using an MCS index m. Let $\delta_{i,k}$ depict a binary indicator variable, capturing whether or not the packet was successfully transmitted and MCS index i was used. Thus, $\delta_{i,k} = 1$ if $i = m$ and the packet's acknowledgment was received and 0 otherwise. Further, suppose that the PSR prediction of the ANN for this packet at MCS index m was P_{mk}. The ADR-X loss function is given by,

$$J_{loss} = \frac{1}{M}\sum_{k}\sum_{i}\delta_{ik}\left(P_{ik} - \delta_{ik}\right)^2 \tag{8}$$

In Eqn. (9) M is the total number of packets. Note the loss function is for the PSR is based on the Eqn. (5) with parameters from the channel model simulations, which will regulate the PSR from being too close to 1 such that the loss is minimized trivially. ADR-X trains the weights of the ANN using gradient decent-based back-propagation to minimize J_{loss} using online training. After each packet, it runs one back-propagation step using the MCS index, success/failure of the latest packet, and the current feature vector $\mathbf{f}$. We use a learning rate of 0.001 in our implementation.

Outline of proof for the validity of ADR-X's loss function. Let p_i be the true PSR for the i^{th} MCS index and q_i be that estimated by ADR-X. As $M \to \infty$, i.e. for a large number of received packets, we have,

$$J_{loss} = p_i\left(1-q_i\right)^2 + \left(1-p_i\right)q_i^2 \tag{9}$$

$$\frac{\partial J_{loss}}{\partial q_i} = 2\left(q_i - p_i\right) \tag{10}$$

Thus, at extremum $q_i = p_i$ i.e. q_i converges to p_i. In other words ADR-X constantly updates the weights to predict the true PSRs.

4.4 Packet Re-transmission Strategy

Re-transmission is used in general to improve packet success rates by giving more chances for the packet to succeed. Packet losses in ADR-X cause two challenges – first, *survivorship bias* and second, *high pre-convergence losses*.

Survivorship bias. When a packet is lost so is the associated CSI and RSS information. This causes survivorship bias i.e.

ADR-X will be able to train only on positive examples using channel measurements and MCS rates from successful transmissions. This can undermine the learning process.

Pre-Convergence losses. In the initial stages of learning or when there are sudden changes in the wireless environment e.g. due to channel change, ADR-X might experience higher losses initially leading to poor audio/video experience.

Packet re-transmission strategy. To solve both these problems, ADR-X relies on a re-transmission strategy of using packet re-transmissions at stepped lower data rate. Upon packet loss, ADR-X re-transmits the packet immediately at three indices below the MCS index suggested by ADR-X for the original transmission. This choice provides insurance by a margin of 10dB prediction error. Optionally, we also use a second re-transmission at four MCS indices below the original transmission in case the first re-transmission is also lost providing a cover of 15dB error. This approach provides negative examples for transmissions that fail as well as positive examples when they succeed in re-transmissions. Further, these re-transmissions also serve as "training wheels" by providing second (or third) chances. As ADR-X learns, its reliance on these "training wheels" diminishes and it succeeds without re-transmissions as we demonstrate in our evaluations.

4.5 Federated Learning for Initialization

When the device is turned on the first time, ADR-X has to start learning from no prior experience. In practice, this means initializing the weights of the neural networks randomly. Starting from no experience, ADR-X may take about 20s - 2 minutes to converge depending on the interference levels in the channel (Section 5). It is however possible to reduce this initial convergence time using two strategies – first, pre-training ADR-X on simulations and second, by leveraging the experiences learned from other consoles using Federated Learning (FL) [32]. ADR-X employs both these strategies in order to reduce initial convergence time.

As we show in our evaluations (Sec. 5.3) using IEEE TGax Channel Model B [67] simulations ADR-X's convergence time reduces to 20-40s with this optimization. The second approach is based on federated learning (FL) algorithms that conduct a *weighted model aggregation* over all the users through weighted averaging of ANN network weights. If $\mathbf{W_m}$ are the weights of the m^{th} device. To fuse the individual models, FL performs a weighted averaging operation:

$$\widetilde{\mathbf{W}} = \sum_{m=1}^{M} \Omega_m \mathbf{W}_m, \tag{11}$$

where Ω_m is the weight matrix of user m for each neuron in the neural networks. For a newly joined user (i.e., a user without any prior experience), we generate and apply a privacy-preserving model [32] with average experience from all users. Since games played on different consoles can be in different environments, the federated model represents a generic model for initialization. However, we note that safe mechanisms to collect data in a privacy-preserving manner may be required to deploy Federated Learning successfully in practice.

4.6 Implementation of ADR-X

We implement ADR-X at the application layer as a PicoScenes Plugin [23], on a desktop or portable laptop equipped with commercial off-the-shelf Qualcomm Atheros 9300 NIC Cards (802.11a/b/g/n) that provides CSI, RSS, NoiseFloor and SNR information for each received packet. We use the PicoScenes toolbox [24] APIs (on Ubuntu 20.04.3.) to collect the CSI measurements for each packet from the NIC. To customize the packet sizes and feedback, we use the packet injection mode provided by the PicoScenes Plugins [23]. PicoScenes APIs allow us to modify packet size, traffic patterns, and MCS directly from the toolbox in real-time. We implemented the ADR-X with about 1.5 K lines of C++ code for the whole ADR-X algorithm, including the gradient descent updates of the neural network.

5 Evaluation of ADR-X

In this section, we evaluate the performance of ADR-X and compare with seven representative state-of-the-art ADR schemes with respect to packet loss rates, consecutive packet loss rates, and average packet transmit times as well as their run times on embedded platforms. We also examine the convergence of ADR-X's learning and the benefit of pre-training through simulations and federated models and the impact of sudden changes in the wireless environment such as channel changes. In the end, we summarize the impact of feature engineering and architecture search for the ANN in ADR-X.

5.1 Experimental Setup

We use the identical setup described in Section 2.1 and implementation in Section 4.6 for conducting our experiments.

Multiple locations and gamers. We tested ADR-X's performance across 10 different gamers at different locations including two different apartments in different apartment complexes and university labs and conference rooms.

Diverse games and traces. We evaluate ADR-X across 20 diverse games, spanning multiple genres such as FPS, racing, and action (list of games with media samples is at https://muralisr.github.io/ADRX/). We use 15-minute (allowing each ADR algorithm sufficient time to converge to a steady state) gaming sessions on each game while recording network packet traces. Each resulting trace contains $> 10^5$ packets.

Embedded devices. Since Xbox devices use 250MHz-600MHz ARM processors, we used Raspberry PI 1 A+ with 250MHz BCM2835 chipset and Raspberry PI 2 600MHz [15] with BCM2836 chipset for evaluating run-times. These embedded devices are representative of the environment requiring link rate adaptation. Specifically, in Xbox, microphone audio transmission occurs from the controller to the console, which is a critical aspect of user experience. Similarly, in the VR scenario, not only is there a comparable audio uplink, but video streaming also occurs from the headset to the server.

Table 1: Summary of the Main Results.

Methods	Channel Info Used	Avg. Total Runtime [μs]	Channel Condition	Xbox					VR				
				Loss Rate [%]			Consective	Average	Loss Rate [%]			Consective	Average
				No Retry	1 Retry	Overall	Loss (>=3)[%]	Tx Time μs	No Retry	1 Retry	Overall	Loss (>=3)[%]	Tx Time [μs]
1. ARF [27] (reactive)	Pkt Loss	0.23	DFS	20.7	4.4	3.7	1.82	376	25.5	5	4.3	2.11	380
			5GHz	25.2	4.7	3.9	1.9	449	27.9	6.2	4.7	2.25	464
			2.4GHz	26.7	5.6	4.9	2.36	476	29.3	6.7	5.3	2.5	482
2. Minstrel [72] (reactive)	Pkt Loss	4.83	DFS	18.8	2.5	1.9	0.63	295	21.9	3.7	3.1	1.2	337
			5GHz	19.6	2.9	2.6	0.84	351	24.9	4	3.5	1.38	360
			2.4GHz	21.8	3.9	3.2	1.02	367	26.2	4.5	3.8	1.57	359
3. RAM [12] (proactive)	SNR	5.62	DFS	8.9	2.3	1.8	0.42	311	11.1	2.2	1.7	0.35	285
			5GHz	10.4	2.8	2.3	0.56	373	13	2.5	1.9	0.44	312
			2.4GHz	11.4	3.4	3	0.68	415	14	2.7	2.2	0.52	337
4. ESNR [18] (proactive)	CSI,SNR	31	DFS	6.5	2.5	2.2	0.4	295	10.5	3	2.4	0.5	280
			5GHz	8.4	3	2.6	0.46	344	12.4	3.2	2.7	0.53	291
			2.4GHz	10.1	3.3	2.7	0.54	395	13.6	3.6	3.1	0.66	312
5. TS [34] (reactive-ML)	Pkt Loss	67	DFS	6.1	2.4	1.9	0.45	281	7.3	2.4	2	0.58	259
			5GHz	7.8	2.7	2.3	0.52	306	9.7	2.6	2.2	0.65	284
			2.4GHz	8.9	3.5	2.9	0.66	325	10.3	3.3	2.6	0.78	303
6. EDRA [11] (proactive-ML)	RSS Pkt Loss	16645	DFS	4.5	2.4	1.6	0.39	264	4.85	2.5	1.4	0.53	238
			5GHz	5.9	2.7	1.8	0.48	279	6.3	2.7	1.5	0.65	270
			2.4GHz	8.1	3.2	2.1	0.62	295	8.4	2.9	2.2	0.73	289
7a. PPO(RSS) (proactive-ML)	CSI,RSS, Pkt Loss	17568	DFS	1.8	0.57	0.13	0.001	248	3.1	0.42	0.15	0.002	225
			5GHz	2.7	0.64	0.21	0.018	257	4.5	0.81	0.43	0.008	239
			2.4GHz	3.9	1.14	0.32	0.021	265	5.3	1.08	0.51	0.015	243
7b. PPO(CSI,RSS) (proactive-ML)	CSI,RSS, Pkt Loss	31081	DFS	0.82	0.23	0.06	0.001	233	0.91	0.15	0.04	0.004	218
			5GHz	1.1	0.31	0.13	0.004	247	1.3	0.26	0.11	0.012	224
			2.4GHz	1.9	0.57	0.21	0.008	266	1.8	0.41	0.17	0.021	239
ADR-X(RSS)	RSS, Pkt Loss	135	DFS	2.4	0.68	0.24	0.015	256	4.8	0.92	0.43	0.024	234
			5GHz	3.5	0.72	0.25	0.023	269	5.4	1.2	0.56	0.038	247
			2.4GHz	5.1	1.71	0.49	0.08	277	6.3	1.9	0.67	0.042	255
ADR-X(CSI, RSS)	CSI,RSS, Pkt Loss	381	DFS	1.2	0.33	0.13	0.003	245	1.5	0.27	0.14	0.007	223
			5GHz	1.4	0.39	0.15	0.01	253	1.7	0.34	0.15	0.014	231
			2.4GHz	2.3	0.79	0.25	0.024	274	2.6	0.56	0.26	0.036	252

This is particularly relevant for scenarios such as VR content sharing and live streaming, underscoring the importance of link adaptation in the embedded device side.

External interference scenarios. In a practical environment, background interference can cause packet losses. Packet losses due to interference occur independent of channel measurements making learning harder for ADR-X. We consider three different interference conditions by operating in three distinct bands – 2.4 GHz band (High Interference), 5 GHz non-DFS bands (Medium Interference), and the DFS bands (No Interference). To provide intuition, Figure 15 depicts the power level and channel width for each of the interfering Wi-Fi devices in the three different Wi-Fi bands within an apartment complex between 10:00-10:15 A.M. at one of the locations where the experiments were conducted. There were 161 active interfering Wi-Fi devices in the 2.4 GHz band, 85 in the 5 GHz band, and none in the DFS band.

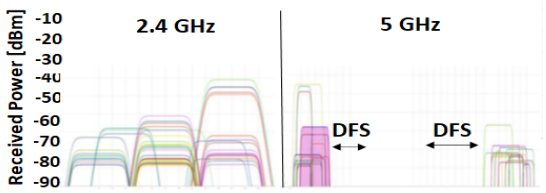

Figure 15: The Wi-Fi environment in 2.4 GHz (High Interference), 5 GHz (Medium Interference) and DFS channels (No Interference).

5.2 Performance of ADR-X

In this section, we show that ADR-X's offers 10× lower loss rate than state-of-the-art under gaming conditions while being computationally efficient. ADR-X is able to achieve this efficiency as it eschews a pure black-box ML approach in favor of synergizing an ANN with communication theoretical results as described in Sec 4.1. This allows ADR-X to sidestep computing analytical expressions for the PSR functions in favor of cheaper switching functions.

Given the large body of prior work in ADR, we choose six representative state-of-the-art ADR schemes from each of the four categories (Sec. 2.2) *reactive*, *proactive*, *reactive using ML* (reactive-ML) and *proactive using ML* (proactive-ML). Further, since Proximal Policy Optimization (PPO) is considered state-of-the-art in reinforcement learning, we implemented our own version of ADR based on PPO as described in Appendix B as an additional comparison point.

Reactive ADR schemes - 1. ARF [27], 2. Minstrel [72]. ARF, found in most off-the-shelf Wi-Fi devices due to its ease of implementation, maintains the packet loss rate as a moving average. The data rate is decreased by one MCS index after two consecutive packet re-transmissions/losses and increased after 9/10 successful transmissions. Minstrel is considered state-of-the-art among reactive ADR schemes and is typically implemented in the Linux driver as the default ADR algorithm. It dedicates 10% of its traffic to probing different rates to search for the maximum achievable throughput.

Proactive ADR schemes - 3. RAM [12], 4. ESNR [18]. Rate

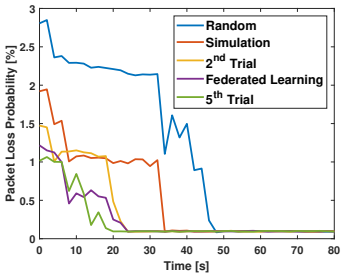

Figure 16: Example of ADR-X(CSI, RSS) convergence in 5GHz for Xbox traffic.

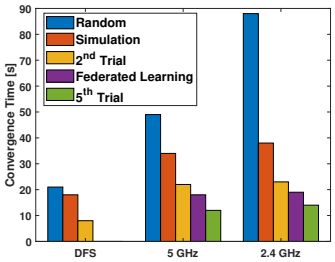

Figure 17: Loss Convergence of of ADR-X(CSI, RSS) for Xbox.

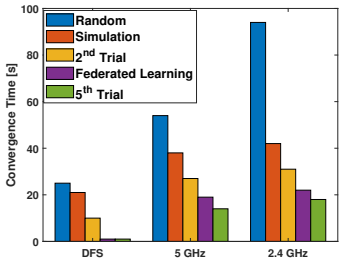

Figure 18: Loss Convergence of of ADR-X(CSI, RSS) for VR.

Adaptation in Mobile environments (RAM) represents the state-of-the-art in proactive schemes that use RSS for channel measurements. RAM maintains a throughput-vs-(rate, SNR) table. It uses measured SNR and the table to select the rate that can maximize throughput. ESNR, described in Section 2.1, represents the state-of-the-art among proactive schemes that use both CSI and RSS. It computes ESNR (Eqn. (3)) using the received packets and computes PSRs to pick the highest data rate that is predicted to have a <10% packet loss rate.

Reactive ADR with ML – 5. Thompson Sampling (TS) [34]. Thompson sampling based ADR is theoretically optimal [51] and considered the state-of-the-art in this category. It takes a multi-armed bandit [37] based on an efficient explore vs exploit approach to selecting the right data rate.

Proactive ADR with ML – 6. EDRA [11] EDRA uses Q-learning based Reinforcement Learning (RL) to select rates aiming to maximize throughput and is the state-of-the-art.

7. PPO based ADR Since, arguably Proximal Policy Optimization (PPO) is the state-of-the-art in RL, we created our own version of ADR that leverages PPO [60] (PPO-Clip) for two cases – i) when only RSS is available as a measurement and ii) both RSS and CSI are available. Due to space constraints, we describe the scheme in detail in the Appendix B. Table 1 summarizes our results averaged across all 10 gamers.

Packet loss rates. For Xbox and VR, and all three interference conditions, only ADR-X and PPO, achieve overall packet loss rates (after two MAC re-transmissions) of 0.1-0.25% with almost negligible consecutive losses – 10× lower than that for all other ADR schemes except ranges around 2-3% (Overall Loss Rate in Table 1).

Loss rates without re-transmissions. To answer the question, "how well does ADR-X pick the data rate compared to other schemes?" we examine the loss rates prior to packet re-transmissions. ADR-X and PPO rely on MAC re-transmissions only 1.2-2.3% of the time (No Retry Loss Rate in Table 1), ≈ 4× improvement over all other schemes that offer between between 6-20%. In fact even with 1 retry, ADR-X and PPO achieve $< 0.5\%$ loss rate while others offer 10× loss rates (1 Retry Loss Rate in Table 1).

Average transmission times. Choosing a rate too low will ensure packet transmission success but increase transmission time. Choosing a rate too high will result in packet losses and incur re-transmissions, once again increasing the overall packet transmission time. Thus, the ability of an ADR scheme to pick just the right rates can be measured by its ability to minimize overall packet transmission time. Packet transmission time is an important metric as it is commonly used as a proxy for energy consumption [62]. ADR schemes that minimize the transmission time also reduce the energy consumption by reducing the duration of time the radio has to be awake for transmission. In Table 1 we compute the average time to transmit a packet. The time to transmit a given packet is the sum of transmission times of all its re-transmissions and the original transmission. As seen from Table 1, ADR-X and PPO offer about ≈40% reduction compared to ARF and ≈25-30% compared to Minstrel and other schemes. An interesting observation is that even ADR-X(RSS) with only RSS, has a $20-25\%$ lower transmission times than all existing schemes.

Average run-time. Table 1 highlights the average run-time per data rate decision by each of the algorithms on computationally constrained embedded chip-sets. The run-time measurements for 250MHz ARM processor are provided in Table 1, and those for the 600MHz processor are provided in Table 2. ADR-X runs ≈ 100× faster than PPO. *It is the only scheme that satisfies the 8-16ms time budget (Sec. 3.1) with significant time left over for duty-cycling to save power.*

Effective Quality of Experience (QoE) hinges on balancing packet losses, transmission time, and power consumption. While a lower MCS reduces packet loss, it may increase transmission time and power usage. High packet loss, especially in Wi-Fi, necessitates retransmissions, further raising power consumption. The primary goal is to finely tune these parameters to minimize packet losses and reduce transmission time, thereby enhancing user experience.

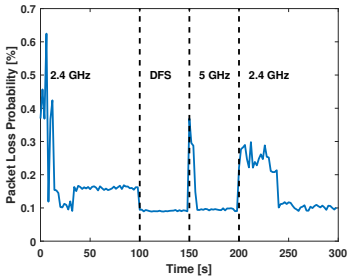

Figure 19: Effect of sudden changes in interference levels.

5.3 Convergence of ADR-X

In this section we ask the question, *"How quickly does ADR-X learn during a gaming session?"*. We define convergence

Table 2: Network Architecture Exploration.

Feature Engineering	Architecture (In-Hidden-Out)	Historical datapoints	Total parameters	Loss (%)	Convergence below 0.5%	Runtime (250 MHz) [μs]			Runtime (600 MHz) [μs]		
						Training	Inference	Total	Training	Inference	Total
Raw (Unsorted)	242-20-1	1	4881	1.31	N/A	4343	3519	7862	1752	1421	3173
	726-20-1	3	14561	0.92	N/A	12511	11515	24026	5348	4928	10276
	726-5-20-1	3	3776	0.13	44s	3496	2708	6204	1403	1107	2510
Sorted	242-20-1	1	4881	1.14	N/A	4250	3408	7658	1761	1426	3187
	726-20-1	3	14561	0.87	N/A	12958	12116	25074	5358	4935	10293
	726-5-20-1	3	3776	0.13	42s	3527	2704	6230	1411	1114	2525
Freq. Binning (Unorted)	27-20-1	3	581	1.34	N/A	478	325	803	194	132	326
	27-5-20-1	3	281	0.76	N/A	206	156	362	84	63	147
Freq. Binning (Sorted)	9-5-1	1	56	0.73	N/A	75	47	122	28	16	44
	9-20-1	1	221	0.65	N/A	182	135	317	73	53	126
	27-5-1	3	146	0.39	89s	134	104	238	55	41	96
	27-20-1	3	581	0.31	96s	461	334	794	196	138	334
	27-5-15-1	3	246	0.21	57s	183	164	347	72	63	135
	27-5-20-1	**3**	**281**	**0.15**	**49s**	**221**	**159**	**381**	**86**	**65**	**151**
	45-5-20-1	5	371	0.17	71s	303	239	542	125	94	219

as the time taken for ADR-X to reach below 0.5% overall loss rate and 0.1% for consecutive losses. We consider four different ANN weight initializations, i) *Random*- randomly initialized ANN weights; ii) Simulation based pre-training - ANN weights obtained by training on simulated CSI, RSS and packet losses using the IEEE TGax Channel Model B [67]; iii) *Federated model* - a federated model (using [32]) based on learned weights obtained from 9 different users across different locations; iv) *Successive trials* - the gamer plays multiple games, each 15 minutes long with a break of 10 minutes in between. After each successive gaming session, weights are passed on to the next session.

To provide an intuition into ADR-X's convergence during a gaming session, Figure 16 depicts the packet loss rates of ADR-X(CSI,RSS) in the 5GHz channel for Xbox traffic. As seen from Figure 16, while random initialization takes about 50s to achieve convergence, simulation-based pre-trained model and federated models take about 35s and 20s respectively. Finally, ADR-X convergence improves to 10s over 5 consecutive gaming sessions.

Figures 17, 18 capture the average convergence times for Xbox and VR over all user experiments under various interference conditions. Interference-free DFS channels are the fastest to converge. The higher the external interference, the longer ADR-X takes to converge. Federated models take about 20s to converge in highly congested channels.

5.4 Sudden Changes in Interference Levels

In this section, we ask the question "How does ADR-X adapt upon experiencing sudden changes in the ambient interference?". This can occur when the radio changes its operating channel into say a highly congested channel. We conduct an experiment starting in a 5GHz channel on a federated model (Sec. 5.3) and then change the operating channel between DFS, 5GHz, and 2.4 GHz as shown in Figure 19. Figures 19 that depicts packet loss as a function of time – each time the channel changes to higher congestion levels ADR-X adapts, however loss rates are below 0.5% at all times.

5.5 Benefits of Feature Engineering

In this section, we answer the question, "How crucial are the various steps of feature engineering to ADR-X?" The feature engineering 4.2 comprises threes aspects – i) use of historical time series of CSI, and RSS values ii) reduction the resolution of the subcarriers through frequency binning and iii) sorting the SNR per subcarriers. Table 2, summarizes performance results for ADR-X with and without different kinds of feature engineering for different ANN architectures. In all cases the including 3 historical channel measurement values brings a significant improvement in ADR-X performance. Frequency binning brings about $10-20\times$ reduction in runtimes over both 250MHz and 600MHz ARM processors. With frequency binning, sorting the SNRs makes a significant improvement in performance – a reduction in loss rates from 0.7% to 0.15%.

5.6 Architecture exploration

We use a Multilayer Perceptron architecture for our ANN. Table 2 presents a few representative examples from our extensive architecture sweep for the ANN architecture – our choice indicated in gray has a loss rate of 0.15% and compute time of 151μs on a 600MHz ARM processor.

6 Conclusions

We perform the first study into wireless losses and their effects on console gaming audio/video experience. Due to gamer hand/head motion during gaming, the wireless channel between a console and its accessories experiences rapid changes. Existing ADR schemes' inability to cope to fast channel changes results in 2-10% losses whereas our study indicates that 0.1-0.5% losses are required for a high quality experience. This paper introduces ADR-X, a contextual ANN assisted multi-armed bandit that closely tracks the channel and adapts quickly. Its novel design exploits communication theory domain knowledge to make it computationally efficient. Overall, ADR-X achieves $10\times$ lower packet loss than the ADR schemes widely used today while running $100\times$ faster than the state-of-the-art ML approaches.

References

[1] Xbox controller teardown. https://www.techinsights.com/blog/xbox-one-teardown.

[2] Milton Abramowitz and Irene A Stegun. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, volume 55. US Government printing office, 1968.

[3] Deepak Agarwal and Bee-Chung Chen. *Statistical Methods for Recommender Systems*. Cambridge University Press, 2016.

[4] Issa Batarseh and S. B. Dewan. *The Switching Function: analysis of power electronic circuits*. The Institution of Engineering and Technology, 2006.

[5] John Charles Bicket. *Bit-rate selection in wireless networks*. PhD thesis, Massachusetts Institute of Technology, 2005.

[6] Michael Biehl and Holm Schwarze. Learning by on-line gradient descent. *Journal of Physics A: Mathematical and general*, 28(3):643, 1995.

[7] Gulnaziye Bingol, Luigi Serreli, Simone Porcu, Alessandro Floris, and Luigi Atzori. The impact of network impairments on the qoe of webrtc applications: A subjective study. In *2022 14th International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6. IEEE, 2022.

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[9] Seongho Byeon, Kangjin Yoon, Changmok Yang, and Sunghyun Choi. Strale: Mobility-aware phy rate and frame aggregation length adaptation in wlans. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, 2017.

[10] Kuan-Ta Chen, Chi-Jui Chang, Chen-Chi Wu, Yu-Chun Chang, and Chin-Laung Lei. Quadrant of euphoria: a crowdsourcing platform for qoe assessment. *IEEE Network*, 24(2):28–35, 2010.

[11] Syuan-Cheng Chen, Chi-Yu Li, and Chui-Hao Chiu. An experience driven design for ieee 802.11ac rate adaptation based on reinforcement learning. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021.

[12] Xi Chen, Prateek Gangwal, and Daji Qiao. Ram: Rate adaptation in mobile environments. *IEEE Transactions on Mobile Computing*, 11(3):464–477, 2012.

[13] Riccardo Crepaldi, Jeongkeun Lee, Raul Etkin, Sung-Ju Lee, and Robin Kravets. Csi-sf: Estimating wireless channel state using csi sampling & fusion. In *2012 Proceedings IEEE INFOCOM*, pages 154–162, 2012.

[14] A Rogier T Donders, Geert JMG Van Der Heijden, Theo Stijnen, and Karel GM Moons. A gentle introduction to imputation of missing values. *Journal of clinical epidemiology*, 59(10):1087–1091, 2006.

[15] Raspberry Pi Foundation. Raspberry pi documentation - computers, 2023. Accessed: [insert date you accessed the site].

[16] Andrea Goldsmith. *Wireless Communications*. Cambridge University Press, 2005.

[17] Rémy Grünblatt, Isabelle Guérin-Lassous, and Olivier Simonin. Simulation and performance evaluation of the intel rate adaptation algorithm. In *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWIM '19, page 27–34, New York, NY, USA, 2019. Association for Computing Machinery.

[18] Daniel Halperin, Wenjun Hu, Anmol Sheth, and David Wetherall. Predictable 802.11 packet delivery from wireless channel measurements. SIGCOMM '10, page 159–170, New York, NY, USA, 2010. Association for Computing Machinery.

[19] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100. AAAI Press, 2016.

[20] Mikael Henaff. Explicit explore-exploit algorithms in continuous state spaces. *Advances in Neural Information Processing Systems*, 32, 2019.

[21] Gavin Holland, Nitin Vaidya, and Paramvir Bahl. A rate-adaptive mac protocol for multi-hop wireless networks. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, MobiCom '01, page 236–251, New York, NY, USA, 2001. Association for Computing Machinery.

[22] Intel. Intel ultimate n wifi link 5300, 2009.

[23] Zhiping Jiang and contributors. Developing Your PicoScenes Plugins. https://ps.zpj.io/plugin.html, 2021.

[24] Zhiping Jiang and contributors. PicoScenes. https://ps.zpj.io/, 2021.

[25] Zhiping Jiang, Tom H. Luan, Xincheng Ren, Dongtao Lv, Han Hao, Jing Wang, Kun Zhao, Wei Xi, Yueshen Xu, and Rui Li. Eliminating the barriers: Demystifying wi-fi baseband design and introducing the picoscenes wi-fi sensing platform. *IEEE Internet of Things Journal*, 9(6):4476–4496, 2022.

[26] Glenn Judd, Xiaohui Wang, and Peter Steenkiste. Efficient channel-aware rate adaptation in dynamic environments. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, MobiSys '08, page 118–131, New York, NY, USA, 2008. Association for Computing Machinery.

[27] Ad Kamerman and Leo Monteban. Wavelan®-ii: A high-performance wireless lan for the unlicensed band. *Bell Labs Technical Journal*, 2(3):118–133, 1997.

[28] Raja Karmakar, Samiran Chattopadhyay, and Sandip Chakraborty. Smartla: Reinforcement learning-based link adaptation for high throughput wireless access networks. *Computer Communications*, 110:1–25, 2017.

[29] Raja Karmakar, Samiran Chattopadhyay, and Sandip Chakraborty. An online learning approach for auto link-configuration in ieee 802.11ac wireless networks. *Computer Networks*, 181:107426, 2020.

[30] Shervin Khastoo, Tim Brecht, and Ali Abedi. Neura: Using neural networks to improve wifi rate adaptation. In *Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '20, page 161–170, New York, NY, USA, 2020. Association for Computing Machinery.

[31] J. Kim, S. Kim, S. Choi, and D. Qiao. Cara: Collision-aware rate adaptation for ieee 802.11 wlans. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–11, 2006.

[32] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[33] Lito Kriara and Mahesh K. Marina. Samplelite: A hybrid approach to 802.11n link adaptation. *SIGCOMM Comput. Commun. Rev.*, 45(2):4–13, apr 2015.

[34] Alexander Krotov, Anton Kiryanov, and Evgeny Khorov. Rate control with spatial reuse for wi-fi 6 dense deployments. *IEEE Access*, 8:168898–168909, 2020.

[35] Mathieu Lacage, Mohammad Hossein Manshaei, and Thierry Turletti. Ieee 802.11 rate adaptation: A practical approach. In *Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '04, page 126–134, New York, NY, USA, 2004. Association for Computing Machinery.

[36] Mathieu Lacage, Mohammad Hossein Manshaei, and Thierry Turletti. Ieee 802.11 rate adaptation: A practical approach. In *Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '04, page 126–134, New York, NY, USA, 2004. Association for Computing Machinery.

[37] T.L Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.

[38] Okhwan Lee, Jihoon Kim, Jongtae Lim, and Sunghyun Choi. Sira: Snr-aware intra-frame rate adaptation. *IEEE Communications Letters*, 19(1):90–93, 2015.

[39] Chi-Yu Li, Syuan-Cheng Chen, Chien-Ting Kuo, and Chui-Hao Chiu. Practical machine learning-based rate adaptation solution for wi-fi nics: Ieee 802.11ac as a case study. *IEEE Transactions on Vehicular Technology*, 69(9):10264–10277, 2020.

[40] Meta. Quest 2. https://www.oculus.com/quest-2/, 2020.

[41] Microsoft. Xbox Series X. https://www.xbox.com/en-US/consoles/xbox-series-x, 2020.

[42] Microsoft. Xbox Wireless Controller. https://www.xbox.com/en-US/accessories/controllers/xbox-wireless-controller, 2020.

[43] Microsoft. Xbox Wireless Headset. https://www.xbox.com/en-US/accessories/headsets/xbox-wireless-headset, 2020.

[44] Babak Naderi and Ross Cutler. An open source implementation of itu-t recommendation p.808 with validation. In *Proc. Interspeech 2020*, pages 1166–1170, 2020.

[45] National Telecommunications and Information Administration. Agreement reached regarding u.s. position, 2003.

[46] Netflix. Vmaf - video multi-method assessment fusion. https://github.com/Netflix/vmaf, 2021.

[47] Ioannis Pefkianakis, Yun Hu, Starsky H.Y. Wong, Hao Yang, and Songwu Lu. Mimo rate adaptation in 802.11n

wireless networks. In *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, MobiCom '10, page 257–268, New York, NY, USA, 2010. Association for Computing Machinery.

[48] Ioannis Pefkianakis, Starsky H.Y. Wong, Hao Yang, Suk-Bok Lee, and Songwu Lu. Toward history-aware robust 802.11 rate adaptation. *IEEE Transactions on Mobile Computing*, 12(3):502–515, 2013.

[49] Ramjee Prasad. *OFDM for Wireless Communications Systems*. Artech House, 2004.

[50] John G Proakis. *Digital communications*. McGraw-Hill, Higher Education, 2008.

[51] Hang Qi, Zhiqun Hu, Xiangming Wen, and Zhaoming Lu. Rate adaptation with thompson sampling in 802.11ac wlan. *IEEE Communications Letters*, 23(10):1888–1892, 2019.

[52] Qualcomm. Qualcomm csrc9300 bluetooth & wi-fi combo chipset, 2014.

[53] Ruben Queiros, Eduardo Nuno Almeida, Helder Fontes, Jose Ruela, and Rui Campos. Wi-fi rate adaptation using a simple deep reinforcement learning approach, 2022.

[54] Hariharan Rahul, Farinaz Edalat, Dina Katabi, and Charles G. Sodini. Frequency-aware rate adaptation and mac protocols. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*, MobiCom '09, page 193–204, New York, NY, USA, 2009. Association for Computing Machinery.

[55] Antony W Rix, John G Beerends, Michael P Hollier, and Andries P Hekstra. Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs. In *2001 IEEE international conference on acoustics, speech, and signal processing. Proceedings (Cat. No. 01CH37221)*, volume 2, pages 749–752. IEEE, 2001.

[56] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly. Opportunistic media access for multirate ad hoc networks. In *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking*, MobiCom '02, page 24–35, New York, NY, USA, 2002. Association for Computing Machinery.

[57] Steven Schmidt, Babak Naderi, Saeed Shafiee Sabet, Saman Zadtootaghaj, and Sebastian Möller. Assessing interactive gaming quality of experience using a crowdsourcing approach. In *2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6. IEEE, 2020.

[58] Steven Schmidt, Saman Zadtootaghaj, Shijie Wang, and Sebastian Möller. Towards the influence of audio quality on gaming quality of experience. In *2021 13th International Conference on Quality of Multimedia Experience (QoMEX)*, pages 169–174. IEEE, 2021.

[59] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[60] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[61] Sayandeep Sen, Neel Kamal Madabhushi, and Suman Banerjee. Scalable WiFi media delivery through adaptive broadcasts. In *7th USENIX Symposium on Networked Systems Design and Implementation (NSDI 10)*, San Jose, CA, April 2010. USENIX Association.

[62] Pablo Serrano, Andres Garcia-Saavedra, Giuseppe Bianchi, Albert Banchs, and Arturo Azcorra. Per-frame energy consumption in 802.11 devices and its implication on modeling and design. *IEEE/ACM Transactions on Networking*, 23(4):1243–1256, 2015.

[63] Gary Smith. *Standard deviations: Flawed assumptions, tortured data, and other ways to lie with statistics*. Abrams, 2014.

[64] Lixing Song and Shaoen Wu. Aarc: Cross-layer wireless rate control driven by fine-grained channel assessment. In *2015 IEEE International Conference on Communications (ICC)*, pages 3311–3316, 2015.

[65] Sony. DualSense Wireless Controller. https://www.playstation.com/en-us/accessories/dualsense-wireless-controller/, 2020.

[66] Sony. PULSE 3D Wireless Headset. https://www.playstation.com/en-us/accessories/pulse-3d-wireless-headset/, 2020.

[67] TGax. Tgax channel model document. Technical report, IEEE 802.11, 2014.

[68] Xiaozheng Tie, Anand Seetharam, Arun Venkataramani, Deepak Ganesan, and Dennis L. Goeckel. Anticipatory wireless bitrate control for blocks. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, New York, NY, USA, 2011. Association for Computing Machinery.

[69] Mythili Vutukuru, Hari Balakrishnan, and Kyle Jamieson. Cross-layer wireless bit rate adaptation. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, page 3–14,

New York, NY, USA, 2009. Association for Computing Machinery.

[70] Abdul Wahab, Nafi Ahmad, and John Schormans. Variation in qoe of passive gaming video streaming for different packet loss ratios. In *2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–4. IEEE, 2020.

[71] Shao-Cheng Wang and Ahmed Helmy. Beware: Background traffic-aware rate adaptation for ieee 802.11. In *2008 International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–12, 2008.

[72] Linux Wireless. Minstrel rate control algorithm. https://wireless.wiki.kernel.org/en/developers/documentation/mac80211/ratecontrol/minstrel, 2015.

[73] Linux Wireless. PID. http://linuxwireless.sipsolutions.net/en/developers/Documentation/mac80211/RateControl/PID/, 2015.

[74] Starsky H. Y. Wong, Hao Yang, Songwu Lu, and Vaduvur Bharghavan. Robust rate adaptation for 802.11 wireless networks. In *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking*, MobiCom '06, page 146–157, New York, NY, USA, 2006. Association for Computing Machinery.

[75] Wei Yin, Peizhao Hu, and Jadwiga Indulska. Rate control in the mac80211 framework: Overview, evaluation and improvements. *Computer Networks*, 81:289–307, 2015.

Appendices

A Deep Reinforcement Learning Design

A General Overview of DRL. Deep reinforcement learning (DRL) is an advanced machine learning approach that combines deep learning and reinforcement learning (RL) techniques. It leverages neural networks to interpret high-dimensional inputs, making it adept at handling complex problems. In the DRL framework, at each time instance t, the RL algorithm necessitates determining the optimal transmission rate for a packet. This decision is based on the input state vector $S(t)$, which incorporates the experience vector harvested from the experience memory. The RL algorithm then prescribes an action state $a(t)$, encompassing the appropriate data rate (MCS) to be employed. Within the RL engine, there exists a mapping from the state to action, denoted as $S(t) \rightarrow a(t)$, which is regarded as the state-action policy map, represented by π_θ. Here, θ symbolizes the parameters of the policy map function, which are updated iteratively during the learning process. Following each transmission, a reward value $r(t)$ is determined to assess the efficacy of selected action $a(t)$ at the current state $S(t)$. These rewards are pivotal in training the RL engine, guiding the optimization of the state-action policy function through the adjustment of θ to enhance the anticipation of future rewards.

State, Action, and Rewards in DRL. The efficacy of a DRL algorithm is fundamentally governed by the appropriate selection and definition of states, actions, and rewards, which are essential in training the DRL model. Initially, we delineate the general design underpinning DRL algorithms, followed by the proposition of two distinct DRL policies sharing identical state, action, and reward designs.

State $s(t)$***:*** The state must ideally comprise all information that is relevant for the actor network and enable it to predict an appropriate action (data rate). In our implementation these comprise channel quality CQ and packet length of the current packet to be transmitted l_t. Since, the most recent channel measurement is usually taken in the past (typically Tms in the past), the channel may have changed within these Tms. The actor must be able to implicitly predict the current state of the channel from history. To enable this, we provide as state, a vector comprising a history of n channel states and lengths. The state is thus computed as $S(t) =< \mathbf{CQ}(t-T), \mathbf{CQ}(t-2T), \cdots, \mathbf{CQ}(t-kT), l_t >$. We use $k = 3$ in our implementation since higher values did not provide any significant benefit. The contents of **CQ** are computed by the Input Generator and depend on the type of channel measurement available in the device. We have designed DRL for three different kinds of channel measurements, each of which uses a different **CQ**.

Action $a(t)$***:*** The action space is all the possible MCSs determined by the IEEE standards for each packet. The policy network outputs the probability estimated by the neural network for actions to achieve the highest reward. Since the action space is discrete, we choose the action with the largest possibility with 95% of the time and 5% of the time we choose the action randomly to explore other rates.

Reward $r(t)$***:*** We design the reward in DRL to specifically target packet loss and power consumption as they are the primary determining factors for user experience in a gaming scenario. In our design we prioritize packet loss more than power consumption since packet loss results in immediate loss of user experience. Further, consecutive packet losses are discouraged to greater degree as they have more significant impact on user experience. The reward function as follows:

$$r(t) = -\tau(MCS) - Q * \rho(t), \tag{12}$$

where $\tau(MCS)$ is the total transmission time by choosing the current MCS, including the retransmission time if the first packet is lost and $\rho(t)$. The term $-\tau(MCS)$ is a penalty on long packet transmission times and hence discourages DRL from choosing low data rates. The term $Q * \rho(t)$ penalizes packet losses with the weight Q determining the trade-off between the power consumption and the packet losses. To prioritize avoiding packet loss $\rho(t)$ is computed as a running average of the packet loss given by :

$$\begin{aligned} \rho(t) &= \rho(t-1) * \alpha + b(t), \\ b(t) &= \begin{cases} 0 & \text{if no packet loss} \\ 1 & \text{if packet loss} \end{cases} \end{aligned} \tag{13}$$

α denotes the reliability requirements, for instance, $\alpha = 0.99$ for the audio packets. In this manner, each packet loss has a long lasting negative impact on the reward, and consecutive packet losses have a greater impact on the reward calculation. This choice discourages DRL to quickly eliminate losses.

B Proximal Policy Optimization (PPO)

B.1 PPO Design

Overview of PPO Algorithm. PPO algorithms belong to class of RL algorithms known an Actor-Critic algorithms comprising two separate neural network models – an Actor and a Critic (Fig. 20). An actor model is a state-action policy map, π_θ to learn what action to take under a particular observed state; θ represent the weights of the neural network. The critic model V_ϕ evaluates the effectiveness of π_θ by predicting the expected future reward based on past history of action-reward pairs;its weights are represented by ϕ. The critic learns by trying to minimize the discrepancy between its past estimates of future rewards and those that it actually observes. The actor learns by attempting to maximize the expected future rewards as predicted by the critic network. Both actor and critic networks learn in conjunction taking turns based on the evolution of state, actions and rewards by employing gradient decent optimization. As time progresses, the critic learns to

Table 3: Network Architecture Exploration for PPO.

Actor		Critic		Loss [%]	Convergence below 0.5%	Runtime (250 MHz) [μs]			Runtime (600 MHz) [μs]		
Architecture (In-Hidden-Out)	Total Parameters	Architecture (In-Hidden-Out)	Total Parameters			Training	Inference	Total	Training	Inference	Total
27-120-9	4320	27-240-1	6720	2.4	NA	13017	4831	17848	5300	2085	7385
		27-40-80-1	4360	2.7	NA	10093	4825	14918	4188	2080	6268
		27-80-40-1	5400	2.8	NA	11531	5003	16534	4657	2087	6744
		27-80-80-1	8640	2.1	NA	14114	5008	19122	6133	2081	8214
27-240-9	8640	27-240-1	6720	0.69	NA	17327	10169	27496	7371	4148	11519
		27-40-80-1	4360	0.98	NA	14900	9660	24560	6239	4115	10354
		27-80-40-1	5400	0.76	NA	16294	10153	26447	6750	4154	10904
		27-80-80-1	8640	0.47	218s	18949	9970	28919	8191	4135	12326
27-40-80-9	5000	27-240-1	6720	0.49	196s	13758	5537	19296	5571	2385	7956
		27-40-80-1	4360	0.65	NA	10346	5773	16119	4454	2406	6860
		27-80-40-1	5400	0.61	NA	12169	5662	17831	4927	2417	7344
		27-80-80-1	8640	0.42	211s	15693	5638	21331	6465	2422	8887
27-80-40-9	5720	27-240-1	6720	0.36	165s	14444	6708	21152	5941	2760	8701
		27-40-80-1	4360	0.43	159s	11708	6659	18367	4844	2766	7610
		27-80-40-1	5400	0.41	162s	12835	6515	19350	5305	2749	8054
		27-80-80-1	8640	0.28	171s	16438	6557	22994	6834	2759	9593
27-80-80-9	9280	27-240-1	6720	0.16	174s	18156	10485	28641	7598	4413	12011
		27-40-80-1	4360	0.24	163s	16210	10586	26796	6561	4417	10978
		27-80-40-1	5400	0.19	169s	16731	11011	27742	7039	4452	11491
		27-80-80-1	8640	0.13	185s	20250	10825	31075	8534	4416	12950

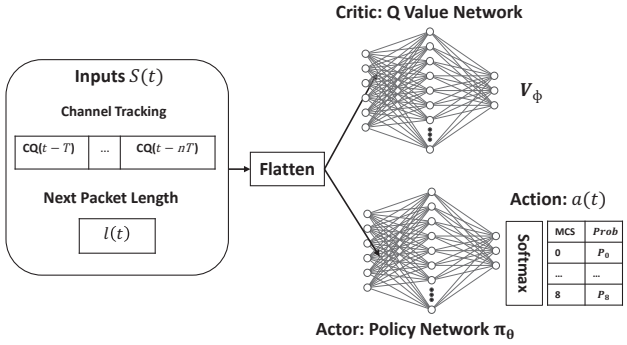

Figure 20: The PPO Architecture for Reinforcement Learning.

predict future rewards more accurately and the actor learns to take more optimal actions for each state. While updating the policy-map, PPO algorithms constrain the amount of change allowed in the policy to limit sudden/drastic changes and hence are stable to sudden changes. In our implementation, we chose a Multilayer Perceptron (MLP) with two hidden layers to represent both actor and critic models

Training Methodology. The objective of the PPO algorithm is to maximize the expected accumulative reward from current time t: $R^\theta(t) = \mathbf{E}_{(S(t),a(t))\sim\pi_\theta}\left[\sum_{j=t}^{\infty}\gamma^{(j-t)}r(j)\right]$, where $\gamma \in [0,1]$ is the discount factor (usually 0.99) used to avoid the accumulated reward to be infinity, and $r(t)$ is the reward by taking action $a(t)$ at state $S(t)$.

The Actor-Critic structure first obtains a finite mini-batch of sequential samples from the trajectory memory. The PPO algorithm randomly chooses a start point within each batch and uses the sub-sequential data to train the network. A new objective function is proposed in PPO to achieve mini-batch updates and update the policy smoothly. PPO introduces importance sampling to obtain the expectation of samples gathered from an old policy π_{old} under the new policy π_{new} we want to refine with the probability ratio $R^\theta(t) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$. They maximize the following surrogate objective function: $L(\theta) = \widehat{E}\left[\min\left(R^\theta(t), \text{clip}\left(R_t(\theta), 1-\varepsilon, 1+\varepsilon\right)\right)\widehat{A}_t\right]$, where ε is the clipping parameter. $\widehat{A}_t$ is an estimator of the advantage function at time step t. We use the generalized advantage estimator (GAE) [59] to calculate $\widehat{A}_t$. By introducing the clipped objective function, the PPO algorithm won't stick to the favoring actions with positive advantage, and make quicker update to avoid actions with a negative advantage function from a mini-batch of samples. $\theta = \theta - \eta_\theta \nabla L^\theta$, where η_θ is the learning rate for the actor model optimization.

Avoiding Losses During Convergence. When PPO is training for the first time or when the environment changes suddenly, PPO may experience higher packet losses during the time it takes to learn, adapt and converge to a steady state. In order to avoid the losses during these vulnerable times, PPO chooses a relies on a conservative retransmission strategy. While RL in PPO provides only the data rate of the initial transmission, the retransmission provides greater reliability by reducing the data rate to an MCS three steps below that suggested by PPO. Note that the corresponding longer transmission time also acts as a discouraging penalty to the reward function

B.2 PPO Network Architecture Exploration.

In this study, we explored various architectural configurations for the actor and critic networks in the PPO algorithm to evaluate their performance on different metrics, including loss

percentage, convergence time, and runtime in two different CPU frequency settings (250 MHz and 600 MHz) for the CSI based input. The architectures were delineated based on the input-hidden-output layers, and each configuration's total number of parameters was reported. A discernible trend is the general improvement in loss percentage and convergence time with more complex network architectures, characterized by a higher number of parameters. The most notable performances were observed in configurations employing the 27-80-80-9 architecture for the actor network, which consistently achieved the lowest loss percentages and reasonable convergence times below 0.5%. Specifically, the combination with the 27-80-80-1 critic architecture exhibited the most promising results, recording the lowest loss of 0.13% and a swift convergence time of 185 seconds.

While the intricate architectures, such as the 27-80-80-9 actor and 27-80-80-1 critic configuration, exhibit good performance in reducing loss and enhancing convergence times, they necessitate significantly extended runtime durations, especially at a frequency setting of 250 MHz. This increase in runtime, which encompasses both the training and inference phases, exhibits a positive correlation with the complexity of the network architectures. In summary, while the PPO network is capable of learning from scratch to find a policy that optimizes the packet loss rate, achieving this necessitates a relatively large network. This complexity poses a challenge for implementation on embedded devices, where computational resources are typically limited. This underscores the need to strike a balance between network complexity and computational efficiency to achieve optimal performance without over-burdening the system resources.