MIA: A Transport-Layer Plugin for Immersive Applications in Millimeter Wave Access Networks

Zongshen Wu*, Chin-Ya Huang† and Parameswaran Ramanathan*
*Department of Electrical and Computer Engineering
University of Wisconsin, Madison, WI, 53706 USA.
†Department of Electronic and Computer Engineering
National Taiwan University of Science and Technology, Taipei, 106 Taiwan.
Email(s): zongshen.wu@wisc.edu, chinya@mail.ntust.edu.tw, parmesh.ramanathan@wisc.edu

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10228878

***Abstract*—The highly directional nature of the millimeter wave (mmWave) beams pose several challenges in using that spectrum for meeting the communication needs of immersive applications. In particular, the mmWave beams are susceptible to misalignments and blockages caused by user movements. As a result, mmWave channels are vulnerable to large fluctuations in quality, which in turn, cause disproportionate degradation in end-to-end performance of Transmission Control Protocol (TCP) based applications. In this paper, we propose a reinforcement learning (RL) integrated transport-layer plugin, *Millimeter wave based Immersive Agent (MIA)*, for immersive content delivery over the mmWave link. MIA uses the RL model to predict mmWave link bandwidth based on the real-time measurement. Then, MIA cooperates with TCP's congestion control scheme to adapt the sending rate in accordance with the predictions of the mmWave bandwidth. To evaluate the effectiveness of the proposed MIA, we conduct experiments using a mmWave augmented immersive testbed and network simulations. The evaluation results show that MIA improves end-to-end immersive performance significantly on both throughput and latency.**

***Index Terms*—mmWave, immersive applications, bandwidth prediction, reinforcement learning, TCP**

I. Introduction

Immersive applications such as augmented reality (AR) and virtual reality (VR) demand high-throughput and low-latency network connectivity to sustain the transmission of rich and dynamic content in their interactive experience. They also need wireless connectivity for untethered access. In this paper, we assume that these connectivity needs of the emerging immersive applications are met using communication over the millimeter wave (mmWave) spectrum.

Since the mmWave beams are highly directional, simple user movements result in beam misalignments and beam blockages, which in turn cause large channel quality fluctuations. Conventional network protocols are not designed to combat the adverse effects of large channel quality fluctuations, thereby causing substantial degradation in end-to-end performance. As a result, immersive applications over conventional network protocols often have poor user experience over mmWave access network.

Immersive applications often use Transmission Control Protocol (TCP) for reliable, sequenced delivery of data. One of the key functions of TCP is congestion control. The variants of TCP such as NewReno [1], Vegas [2], BIC [3], CUBIC [4], Compound [5], BBR [6] and PCC [7] differ primarily in their congestion control schemes. These variants are adept in dealing with congestion caused by increased traffic demand on a wired bottleneck link, but they are not adept in dealing with congestion or even rapid changes of the bottleneck bandwidth caused by channel quality fluctuations of a wireless link. To combat the challenges posed by wireless links, researchers have proposed a wide range of solutions, ranging from the pioneering work by Balakrishanna *et al.* in Split-TCP [8] to the more recent [9]; the number of proposed solutions are too numerous to explicitly cite here. A large fraction of the work in literature on wireless related TCP solutions focus on cellular network where the data rates are relatively small. When the data rates of the wireless links are large, as in mmWave networks, the underlying network typically has large bandwidth-delay product, which in turn introduces new challenges to TCP. Specifically, the sending rate adaptation at the TCP sender is often not available to react quick enough to deal with large fluctuations in the data rate of a bottleneck link. Recent works that focus on this situation are [10]–[12]. These solutions rely on dual or multi connectivity in the underlying network to combat the effects of large channel quality fluctuations of a single wireless link. Although dual or multi connectivity is certainly possible in wireless networks, they are still less common than network with single connectivity. In this paper, we focus on single connected networks, where one does not have the option to make use of hopefully another better connected path in the network.

In this paper, we design a novel reinforcement learning (RL) integrated transport-layer plugin, Millimeter wave based Immersive Agent (MIA), to specifically combat the deleterious effects of mmWave's large channel quality fluctuations. Inspired by the approach in DeepCC [13], this plugin works in conjunction with any throughput-oriented congestion control scheme of TCP to deliver superior end-to-end performance to the application. As in many recent works [13]–[15], this plugin takes advantage of recent advances in RL. In particular, it uses a RL scheme called Deep Q Network (DQN) to predict the quality of the mmWave channel at the receiver end of the TCP connection. This prediction is fed back to the sender end of the TCP connection through acknowledgements (ACK), where it is utilized along with TCP's congestion control scheme to

make the transport-layer sending rate cognizant of potential changes in the wireless link bandwidth. Simulation results included in the paper show that there are two to six fold increase in the end-to-end throughput as compared to that in different TCP variants. The simulation results also show a considerable decrease in the end-to-end latency delivered to the immersive applications.

Contributions: The contributions of this paper are as follows:

1) We provide a comprehensive system design for MIA plugin, which includes the functionality of prediction, controlling and measurement.
2) We provide a DQN based model which utilizes the user speed information and demonstrate its effectiveness in mmWave link bandwidth prediction.
3) We build a real-time measurement testbed which traces the mmWave link bandwidth along with the VR user speed information in immersive games.
4) We demonstrate that, with different network delays and games, MIA plugin provides multi-fold improvement to different variants of TCP.

The rest of this paper is organized as follows. The related work is introduced in Section II. The proposed approach is described in Section III. The evaluation setup and results are presented in Section IV. Finally, the paper is summarized in Section V.

II. Related Work

Machine Learning. The machine learning literature is clearly vast. Here we only review the couple of machine learning approaches that are directly relevant to the proposed work. Deep Q Network (DQN) [16] and Deep Deterministic Policy Gradient (DDPG) [17] are two well-known RL algorithms. They are both well-suited for adaptively learning from high-dimensional inputs and arriving at simple low-dimensional decisions/actions. In the aspect of RL's empirical learning structure, DQN and DDPG have the similar RL structure. That is, they both train based on the states, actions and the corresponded rewards. And their evaluation theorems are both based on Bellman equation. The main difference between DQN and DDPG in applications is that DQN supports discrete actions while DDPG supports continuous actions.

Machine Learning for TCP. TCP has shown its potential in cooperating with machine learning algorithms in studies. Remy [18] is a pioneering work showing that machine learning algorithm can be used for generating congestion control rules with network features. Remy generates the mapping rules with the knowledge of the network assumptions, traffic model and object functions for the fixed network model. [19] uses decision tree to detect the packet loss in wireless connection during congestion control after trained with the knowledge of network features. [20] use XGBoost classifier to emulate the behaviors of different TCP congestion control schemes. These machine learning based congestion control scheme designs require the presumption of the networks. When the network condition becomes dynamic and difficult to be predefined for specific training, RL becomes a competitive candidate to adapt to different network conditions. Owl [15] uses DQN to learn the network features of partially invisible networks and select proper congestion window with partial network knowledge. [21] states the potential overhead when applying RL to congestion control schemes in unseen networks. DeepCC [13] uses DDPG for TCP congestion control in celluar network in order to steer TCP from throughput oriented toward delay oriented. The architecture of DeepCC is compatible with different congestion control schemes, which inspires the plugin design of MIA for immersive applications in mmWave access networks. DRL-CC [14] applies DDPG with Long Short-Term Memory (LSTM) [22] representation network to improve the congestion control in MPTCP when the number of TCP subflows is various. DRL-CC uses LSTM to control multiple MPTCP subflows when the flow number may change over time. As for the control on each TCP flow, DRL-CC uses DDPG as the RL algorithm for the congestion control. The target of DRL-CC is achieving higher TCP throughput and lower file transferring latency, which aligns with our target in the mmWave based immersive applications. With the same target, when further applied to mmWave based immersive applications, DRL-CC can cooperate with MIA to overcome the potential difficulties. DRL-CC's empirical congestion control for each TCP flow is still only based on the observation at the sender, which may not react to the rapid mmWave link bandwidth changes in real time. Meanwhile, observing and training at the sender limit DRL-CC from extracting extensive features in the mmWave based immersive environment, where the mmWave link is the fluctuated bottleneck under the impact of the immersive user.

mmWave Related Work. Challenges of mmWave communications have been addressed at all layers of network stack. At physical layer, Sur *et al.* propose MUST to predict the best beam and to redirect user traffic over conventional WiFi when there is blockage in the mmWave link [23]. Although MUST focuses on the physical layer, it needs support from higher layers of the protocol to redirect user traffic. [10] discusses the challenges and tradeoffs of Ultra-Reliable Low Latency Communication (URLLC) considering massive MIMO and multi-connectivity.

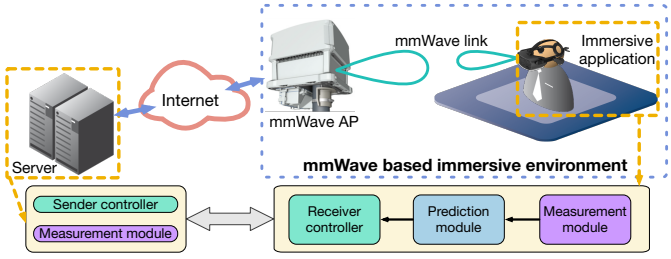


Fig. 1: Overview of MIA structure.

III. Millimeter Wave Based Immersive Agent (MIA) Scheme

A. Overview

For simplicity of presentation, Fig. 1 shows a simple topology formed by an immersive content server, a mmWave

access point (AP), and a mobile immersive device worn by the user. The mobile immersive device is enhanced with mmWave access to the AP and the immersive content is sent from the server to the mobile immersive device in an immersive application. In the immersive experience, the user with the mobile device moves in a small geographic area in the vicinity of the mmWave AP. Due to the highly directional feature of mmWave beam, the mmWave channel quality is vulnerable to misalignment and blockages. Therefore, when the user induces frequent changes in location and direction during the interactions with immersive applications, the mmWave channel quality fluctuates dramatically [24]. Consequently, during the immersive experience in the mmWave access network in Fig. 1, the bandwidth bottleneck on the route from the server to the mobile device is often at the wireless mmWave link between the AP and the immersive mobile device.

We propose Millimeter wave based Immersive Agent (MIA), a RL integrated transport-layer plugin, to adaptively adjust the transport-layer packet forwarding rate for better performance. Shown as Fig. 1, MIA consists of three modules, the prediction module, the measurement module, and the control module (i.e., sender controller and receiver controller). The prediction module is implemented at the immersive mobile device, which is the receiver of the immersive content. The control module and the measurement module are implemented at both sender and receiver sides. MIA's prediction module takes the user movements and the current mmWave link bandwidth as the inputs and predicts the mmWave link bandwidth in the near future, that is, at the next round trip time (RTT). The input data are collected by the measurement module and the control module adaptively adjust the data sending rate with the predicted link bandwidth.

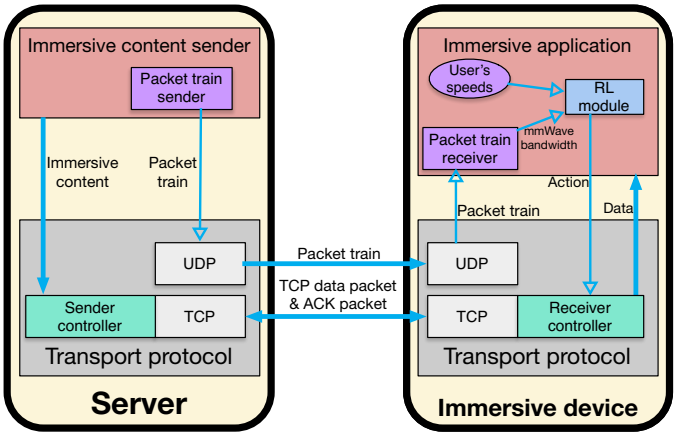


Fig. 2: MIA system architecture.

B. MIA System Architecture

The system architecture of MIA is presented in Fig. 2. MIA includes the cross-layer design between the immersive application and the transport layer protocol. The immersive content is sent from the remote server to the application at the immersive device. At the server, the immersive content sender forks the packet train sender in the application and forwards

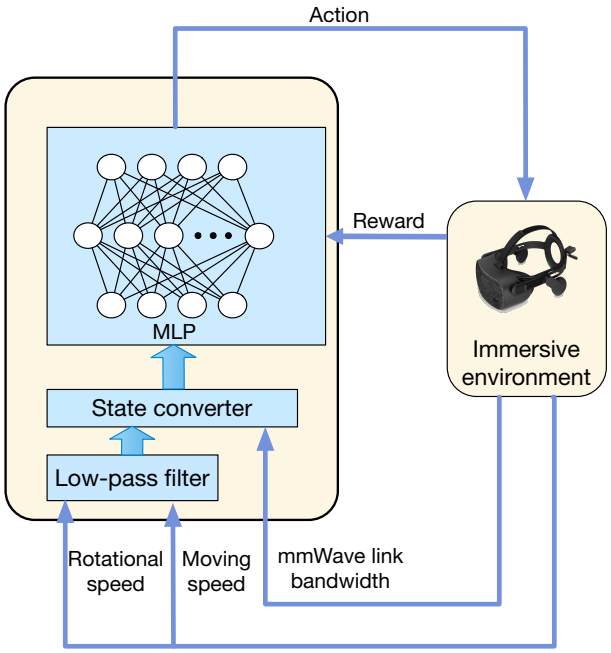


Fig. 3: RL module.

immersive content to the sender controller. The packet train sender operates as the sender of the measurement module and sends packet trains to the immersive device for link bandwidth measurement through the UDP connection. The sender controller is integrated with TCP at the server and the TCP sends data packets containing immersive content to the immersive device. The control scheme at the server uses the information extracted from the ACK packet arriving at the server in MIA. At the immersive device, the packet train receiver is forked by the immersive application. It calculates the mmWave link bandwidth according to the incoming packet trains from the UDP connection and forwards the measured mmWave link bandwidth to the RL module in the application. The user's speed information is also sampled in the immersive application during the user's immersive experience. Shared memory is used to send the mmWave link bandwidth and the speed information from the measurement module to the RL module. The RL module takes the user's speed information and mmWave link bandwidth as inputs and outputs the action to the receiver controller through shared memory. The TCP with the receiver controller forwards the received immersive content up to the application. Meanwhile, the receiver controller takes the action from the RL module and integrates this action information with the ACK packet sent from the immersive device to the server. Then, the server further controls its sending process with this information. Note that, in MIA system architecture, the prediction, control and measurement modules run in parallel. The server runs the packet train sender and the sender controller simultaneously since there is no interaction between them. The immersive device runs its measurement module, RL module and receiver controller in parallel with shared memory connecting these modules. In this case, there is no blocking among these modules. This non-

blocking architecture avoids potential latency overhead when applying RL module to TCP.

Algorithm 1 DQN prediction module in **MIA**.

Training process of one VR game:

Initialize replay buffer B with capacity of N_B
Initialize state queue S with capacity of N_S
Initialize action-value network Q with random weights θ
Initialize target action-value network $\hat{Q}$ with weights $\hat{\theta} = \theta$
for episode = $1, ..., M$ **do**
  Reset the timestamp to the start of the VR experience.
  for t = $(1, ..., T) \cdot \tau$ **do**
    Sample user moving speed value $v_{m,t}$, rotational speed value $v_{r,t}$, and mmWave link bandwidth value $v_{b,t}$
    Process $v_{m,t}$, $v_{r,t}$ with low pass filter and get $v'_{m,t}, v'_{r,t}$
    Convert input values to state: $s_t = [s_{b,t}, s_{m,t}, s_{r,t}] = StateConverter(v_{b,t}, v'_{m,t}, v'_{r,t})$
    Enqueue state s_t, $S_t = [s_t, s_{t-1}, ..., s_{t-N_S+1}]$
    if Probability falls in ϵ **then**
      Take random action a_t
    else
      Take action $a_t = \underset{a}{\operatorname{argmax}}\, Q_\theta(S_t, a)$
    end if
    Execute action a_t in VR channel environment and collect reward r_t and state S_{t+RTT}
    Record the transition $(S_t, a_t, r_t, S_{t+RTT})$ into replay buffer B
    Randomly sampled m transitions from buffer B, noted as at time t'
    if VR game ends at $t' + RTT$ **then**
      Target value $y_{t'} = r_{t'}$
    else
      Target value $y_{t'} = r_{t'} + \gamma \cdot \max_a \hat{Q}_{\hat{\theta}}(S_{t'+RTT}, a)$
    end if
    Calculate the Huber loss $L_\delta(y_{t'}, Q_\theta(S_{t'}, a_{t'}))$
    Optimize weights θ with gradient descent
    Update the target network weights $\hat{\theta} = \theta$ for every C epochs
  end for
end for

C. mmWave Link Bandwidth Prediction Module

The structure of the prediction module is presented in Fig. 3. The key point of this module is to predict the mmWave link bandwidth at the near future with the observation of user movements and corresponded link bandwidth. In our design, the prediction module is implemented at the user device, which is the receiver of the immersive data. Thus, the prediction at the user device represents the mmWave link bandwidth at the next RTT. We use DQN [16] with multilayer perceptron (MLP) policy in our RL module. Compared with transition table, DQN supports more (input, action) transitions, while numerous transitions require large transition tables. Moreover, we quantize the input data and actions into discrete states and use single agent for prediction. Compared with other RL algorithm variants, DQN is deterministic and it fits our immersive scenario where the action is discrete and low-dimension.

The DQN module takes user's current moving speed, rotational speed, and mmWave link bandwidth as inputs and converts these input values into states. The action of the DQN module is also generated in states and then converted to action values as the output. The DQN interacts with the mmWave immersive environment and gets the feedback through the reward calculated by the environment. The design of the state, action and reward function is described as below.

STATE: At time t, the state s_t includes $s_{b,t}, s_{m,t}, s_{r,t}$, which represent the current mmWave link bandwidth state, user moving speed state, and user rotational speed state. To generate these state values, the prediction module first samples mmWave link bandwidth value $v_{b,t}$, user moving speed value $v_{m,t}$, and user rotational speed value $v_{r,t}$. Then, we use Butterworth filter to filter out the high frequency noise in $v_{m,t}$ and $v_{r,t}$ during the immersive experience. The filtered values of moving speed and rotational speed are represented as $v'_{m,t}$ and $v'_{r,t}$. Next, we put $[v_{b,t}, v'_{m,t}, v'_{r,t}]$ into a state converter, which converts these values into states $s_{b,t}, s_{m,t}, s_{r,t}$. These states form into s_t. We select the user moving speed and rotational speed in the state construction because they describe the changes in location and direction, which impact the mmWave link bandwidth. The current state s_t is further enqueued into the state queue S, which is the state observed in the DQN algorithm. The capacity of S is N_S, which means the input size of the DQN is $3 \cdot N_S$.

ACTION: The action a_t represents the prediction of mmWave link bandwidth at the next RTT. Due to the action in DQN is discrete, the final output of the prediction is mapped from the action state to the corresponded bandwidth value.

REWARD: The reward function is designed to show each prediction's accuracy compared with the true mmWave link bandwidth. Assume at time t, the prediction a_t is generated for the next RTT. Then, the prediction error is defined as:

$$e_t = \frac{a_t - v_{b,t+RTT}}{v_{b,t+RTT}} \tag{1}$$

The corresponded reward is:

$$r_t = \begin{cases} \alpha \cdot \exp(-0.5 \cdot (\frac{e_t}{\rho_s})^2) + \beta & \text{if } e_t > 0 \\ \alpha \cdot \exp(-0.5 \cdot (\frac{e_t}{\rho_l})^2) + \beta & \text{otherwise} \end{cases} \tag{2}$$

The scale and offset of the reward function is adjusted by α and β. When the prediction is equal to the ground truth, the reward reaches the maximum value. As the prediction error increases, the reward drops and converges to the minimum value. We usually set ρ_l larger than ρ_s. We use larger descending rate when the prediction is greater than the ground truth because when larger prediction leads the sender sending

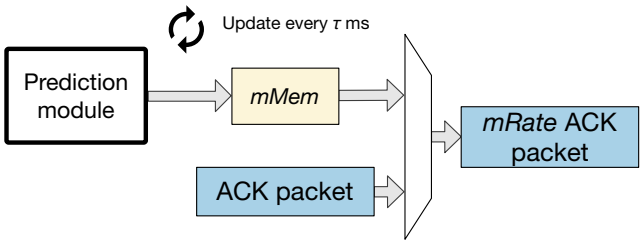


Fig. 4: Control scheme at the receiver controller.

too many packets, it may result in buffer overflow at the AP and harm the performance significantly.

Algorithm 1 describes the training process of our DQN algorithm in detail. At the start of the algorithm, the module initializes the replay buffer B and state queue S with capacity of N_B and N_S separately. We use a three-layer MLP with 32 neurons on each layer as the action-value network Q. The hidden layers are activated by rectified linear activation function and the last hidden layer is fully connected with the actions on the output layer. The target network $\hat{Q}$ has the same structure as Q. The network Q is initialized with random weights and $\hat{Q}$ is assigned with same initial weights as Q. The DQN is trained for M episodes. Each episode means one VR game experience. During the training, the algorithm takes each epoch every τ milliseconds. We use t to represent the timestamp of each epoch. Within each epoch, the algorithm first converts the input data into states and enqueues these states (line 8-11). When choosing the optimal action, the algorithm takes the action which has the maximum Q-value calculated by the DQN (line 15). Meanwhile, to comprehensively explore possible transitions, the algorithm takes a random action with a probability of ϵ (line 13). After execute action a_t in environment. The transition $(S_t, a_t, r_t, S_{t+RTT})$ is stored into replay buffer B. The replay buffer is used for storing transitions during the training process and the DQN can randomly samples stored transitions from B in mini-batch mode. The sampled transitions are used to calculate the target value through target network $\hat{Q}$ with Bellman equation (line 23). Then, we calculate the Huber loss and use Adam optimizer [25] to train the DQN. Note that, the weights of $\hat{Q}$ is updated every C epochs with the weights of Q for stability during training.

D. Control Scheme

1) Control Scheme at Receiver Controller: Fig. 4 describes the control scheme at the receiver controller, which cooperates with TCP. The receiver controller takes the action value from the prediction module in the immersive application through a shared memory, named as $mMem$. The $mMem$ is updated every τ milliseconds. This aligns with the time when the prediction module takes an action. When the receiver sends out an ACK packet, it appends the value of $mMem$ to the ACK packet, noted as $mRate$. The $mRate$ represents the sending rate adapted to the predicted mmWave link bandwidth. Then, the sender executes further controls based on $mRate$.

Algorithm 2 Control scheme at the **sender** controller.

Initialize threshold TH_{MIA}

For each ACK packet receive process:

1: Extract $mRate$ as predicted mmWave bandwidth limit
2: Get current time t_{now}
3: Calculate the moving average of sending intervals: $t_{int} = t_{now} - t_{stamp}$
4: **if** ACK packet is duplicate **then**
5: Update counter: cnt = 0
6: **else**
7: Update counter: cnt += 1
8: **end if**
9: Calculate rate adaptive window: $mWnd = mRate \cdot t_{int}$
10: **if** $mWnd < \min(cWnd, rWnd) - B_f$ **then**
11: $sWnd = mWnd$ ▷ rate adaptive mode
12: **else**
13: **if** $cnt > TH_{MIA}$ **then**
14: $sWnd = mWnd$ ▷ rate adaptive mode
15: **else**
16: $sWnd = \min(cWnd, rWnd) - B_f$ ▷ congestion mode
17: **end if**
18: **end if**
19: Send $sWnd$ bytes of data
20: Record the current time as a timestamp: $t_{stamp} = t_{now}$

2) Control Scheme at Sender Controller: Algorithm 2 describes the control scheme at the sender controller. The controller is first initialized with the threshold value TH_{MIA}. This threshold is used to determine whether to turn on the rate adaptive mode in MIA. When an ACK packet arrives at the sender, the sender controller extracts the $mRate$ as the predicted bandwidth limit. Meanwhile, the current timestamp t_{now} is updated and the arriving packet interval t_{int} is calculated as the time duration for each sending process. The t_{stamp} represents the timestamp at the previous sending process. Moving average is used for t_{int} to alleviate abrupt large changes in packet intervals when the mmWave link bandwidth fluctuates dramatically. Then, the counter cnt is updated according to the ACK packet (line 4-8). The counter cnt records the number of consecutive new ACK packets, which is further used for mode choosing. During the mode choosing (line 9-18), the $mWnd$ is first calculated as the sending limitation with $mRate$ and t_{int}. The mode is decided through two comparisons. The first comparison is based on $mWnd$, congestion window $cWnd$, receive window $rWnd$ and bytes in flight B_f. The second comparison is between the counter cnt and TH_{MIA}. In rate adaptive mode, the sending limit window $sWnd$ is equal to $mWnd$. Otherwise, $sWnd$ is controlled by $cWnd$, $rWnd$, and B_f. After setting the sending limit window $sWnd$, the sender sends $sWnd$ bytes of data. Finally, the timestamp t_{stamp} is updated for the future packet interval calculation.

E. Low-Overhead Real-Time Measurement Module

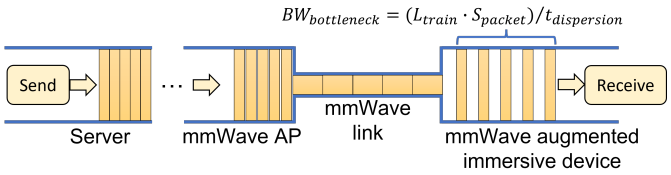


Fig. 5: Illustration of the packet train.

The measurement module provides real-time input data to the prediction module in MIA, including user's moving speed, rotational speed and mmWave link bandwidth. The speed information can be measured from immersive devices with little overhead. The key point is accurately measuring mmWave link bandwidth with low overhead. MIA applies packet trains [26] for the accurate estimation of the mmWave link bandwidth with low bandwidth consumption. A packet train is one group of packets with the same packet size. In each train, the packets are generated and forwarded back-to-back by the packet train sender in the server's application. This packet train is then received by the immersive device over the mmWave link. As illustrated in Fig. 5, this technique relies on the observation that these packets are usually queued and sent in an back-to-back manner. If this observation holds, the time spaces between these packets (i.e., time dispersion) are inversely proportional to the bandwidth of the bottleneck link on the route from the sender to the receiver, which is the mmWave link in our immersive scenario. If L_{train} is the length of one random section in a packet train and the packet size is S_{packet}, then the relationship to bottleneck bandwidth in this section can be shown as:

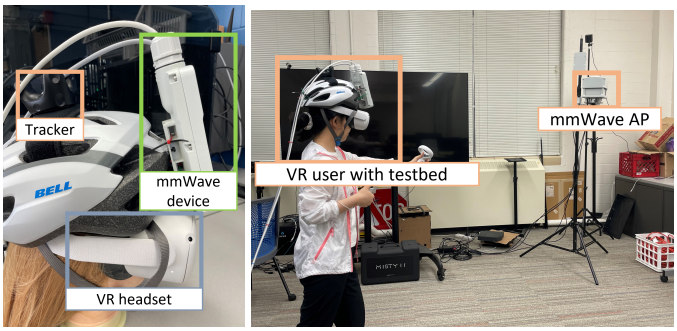
$$BW_{bottleneck} = (L_{train} \cdot S_{packet})/t_{dispersion} \quad (3)$$

When receiving a packet train, the receiver randomly samples different consecutive sections from this packet train and takes the average measurement result of these sections. In the design of MIA, concatenated with prediction module, the interval between two packet trains is τ milliseconds. This aligns with the prediction frequency in Sec. III-C and the $mMem$ updating frequency in Sec. III-D.

IV. Evaluation

A. Experimental Setup

We first build up a real-time measurement testbed to measure the user movements and the bottleneck link bandwidth while the user is playing a VR game (see Fig. 6). In the testbed, the user plays immersive games in a 5m x 5m space in the vicinity of the mmWave AP. Since the current Oculus Quest 2 headsets do not support mmWave connections, the testbed augments them with IEEE 802.11ay [27] mmWave devices. In the testbed, the bottleneck link is often the mmWave link and therefore its bandwidth fluctuations are measured using the packet train technique described in Sec. III-E. The user's moving and rotational speeds are measured using a HTC Vive tracker, which is attached to the helmet. We collect data and evaluate performance in three immersive game scenarios. The immersive experience lasts for approximately 20 minutes in each game scenario. The trace data from the measurements is used in an implementation of MIA in Network Simulator-3 (ns-3) [28] along with ns3gym [29] and OpenAI Gym [30].



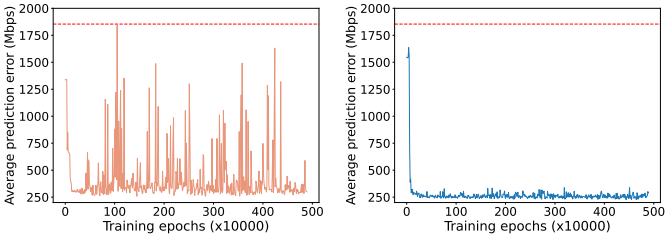
(a) Testbed. (b) A user in immersive experience.

Fig. 6: MIA setup.

We evaluate the performance of the RL prediction module and then evaluate MIA's performance when working with different congestion control schemes implemented in four variants of TCP, Namely TCP-BIC [3], TCP-CUBIC [4], TCP-BBR [6], and DRL-CC [14]. The effectiveness of the MIA is assessed by comparing the performance of a given congestion control scheme without and with the proposed MIA.

TABLE I: Hyperparameters of learning module.

Hyperparameter	Variable	Value
Replay buffer capacity	N_B	1000000
State queue capacity	N_S	10
Episode	M	500
Epoch time	τ	10ms
Exploration rate	ϵ	1 – 0.05
Minibatch size	m	32
Target network update interval	C	10000
Reward function scale	α	10.0
Reward function offset	β	-4.0
Reward function deviation (large)	ρ_l	0.25
Reward function deviation (small)	ρ_s	0.1



(a) Training without user speed information. (b) Training with user speed information.

Fig. 7: Comparison of average link bandwidth prediction error without and with using user movement information while the user is playing a boxing game over a network with an average delay of around 10ms.

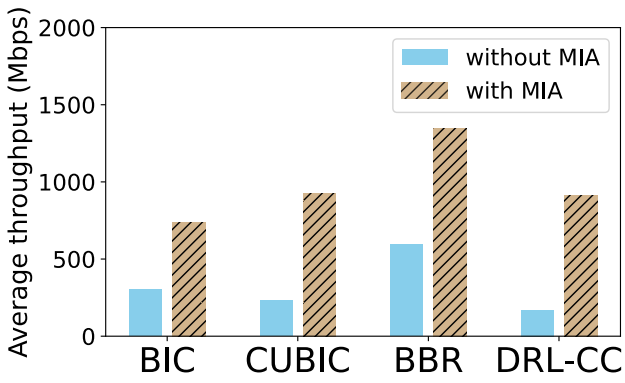


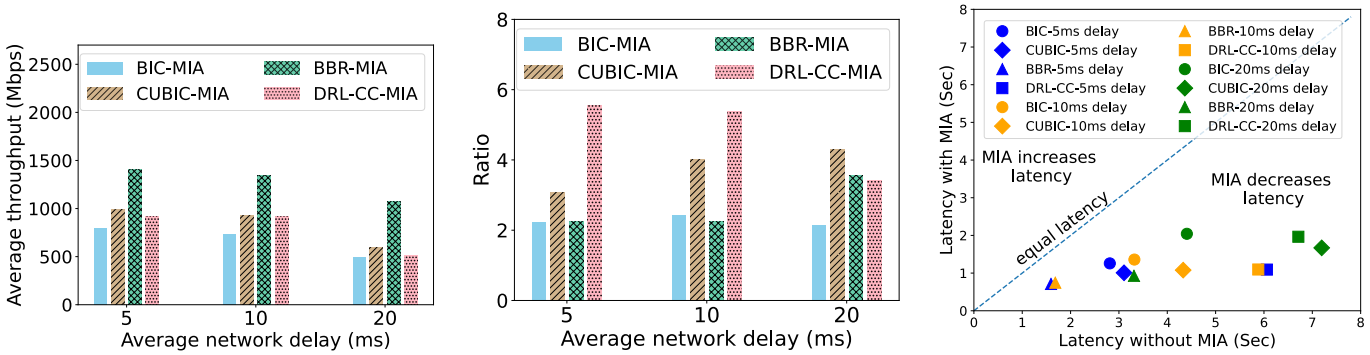
Fig. 8: Average throughput with 10ms average network delay when different TCP variants run without and with MIA in an boxing game.

B. Experimental Results

1) Prediction Module: We compare the convergence of two different approaches for training the RL model. In the first approach, only a recent history of data rates is used. In the second approach, a recent history of user's moving and rotational speeds are also used along with a recent history of data rate fluctuations. The comparison is used to evaluate and demonstrate the benefits of using recent history of user movements in predicting future data rates. Table. I shows the hyperparameters of the RL model during the training process. We train and test different models for three different immersive games (boxing, beat saber, and table tennis) and three average network delays (5ms, 10ms, 20ms). Each model is trained for 5 million epochs with an immersive environment of 20-minute gaming experience. Then, the model is tested in the immersive environment of another 20-minute gaming experience when the user plays the same game on a different day and time. Fig. 7 shows the absolute values of average prediction errors in mmWave link bandwidth during the training process of the model in the boxing game with 10ms average network delay, corresponding to an average RTT of around 20ms. The red dashed line represents the average error in one episode when the model takes random actions. Fig. 7a shows the average errors when the model **only** takes mmWave link bandwidth as the input. As the comparison, Fig. 7b shows the average errors when the model also takes the user's moving speed and rotational speed as input. The comparison shows that the model with speed information is much better. The prediction errors are much smaller and training converges rapidly. Therefore, we use the learning module applying user's moving speed and rotational speed for the following performance evaluation and we use the weights at the end of the training process for the prediction module.

2) Performance Improvement due to MIA: We integrate MIA with different TCP variants, including TCP-BIC, TCP-CUBIC, TCP-BBR and DRL-CC. Then we evaluate their performance with different immersive gaming experience in simulations done using ns-3. In these simulations, the network between the server and the user device is comprised of three communication links. The first two links are assumed to be wired with a bandwidth of 10 Gbps, while the third is a mmWave based wireless link whose bandwidth varies based on the traces collected by experimental measurements. In order to model a typical network, we pick a target value for RTT. We set the propagation delay for the first communication link to be normally distributed with mean equal to half of the target RTT and variance equal to $\frac{1}{20}$ of the target RTT. The second communication link is also shared by an UDP cross-traffic with an average bandwidth of 2 Gbps. The cross-traffic may cause congestion on the second link. The queuing in the second communication link also causes variability in the RTT. Finally, the bandwidth changes in the wireless link also causes considerable variations to the delays experienced by each packet. First, we present simulation results for the boxing game when the average network delay is around 10ms (see Fig. 8). The figure compares the average end-to-end throughput over the 20-minute game for four different TCP variants without and with the MIA. For instance, DRL-CC without MIA has an average throughput of only 170 Mbps while the DRL-CC with MIA has an average throughput of 913 Mbps, a 5.3-fold increase in the throughput. Since DRL-CC is also a reinforcement learning based approach this improvement in performance is noteworthy. We therefore contend that link bandwidth prediction at the receiver end is very effective in improving the end-to-end throughput. This is understandable because link bandwidth prediction allows TCP to adapt its sending rate well in advance of changes in channel condition. In particular, reducing sending rate in advance of expected poor channel conditions is very advantageous in reducing the queueing delays and packet losses at the wireless link. Furthermore, when the channel conditions are expected to be good and there are no hints of network congestion elsewhere as indicated by the continuous arrival of new acknowledgements, the sending rate can be increased more rapidly to accommodate the additional availability in the network. Note that, although there are differences in improvement based on the underlying congestion control scheme, the gains are substantial for all four TCP variants.

Next, in Fig. 9, we further evaluate the improvements provided by MIA. In Fig. 9a, we show the impact of network delay (and hence, RTT) on the end-to-end throughput. As shown in the figure, the average network delays are 5ms, 10ms, and 20ms. The four bars for each network delay correspond to the four TCP variants used in conjunction with MIA. The figure basically shows that, as expected, the throughput of all four variants decreases with increase in delay. The throughput of BBR-MIA decreases from around 1400 Mbps to around 1000 Mbps as the delay increases from 5 ms to 20 ms. Similarly, the throughput of DRL-CC-MIA decreases from around 900 Mbps for 5 ms delay to around 500 Mbps for 20 ms. The corresponding Fig. 9b shows the relative improvement in end-to-end throughput with MIA and without MIA. Irrespective of the network delays, the throughput improvements range from a factor of 2 to a factor of nearly 6. A 5.5-fold increase in average end throughput for 5 ms delay for DRL-



(a) Average throughput when MIA cooperates with different TCP variants.

(b) Improvement ratio when MIA cooperates with different TCP variants.

(c) Data transmission latency comparison between using and not using MIA.

Fig. 9: Performance metrics in the boxing game when MIA cooperates with different network delays and TCP variants.

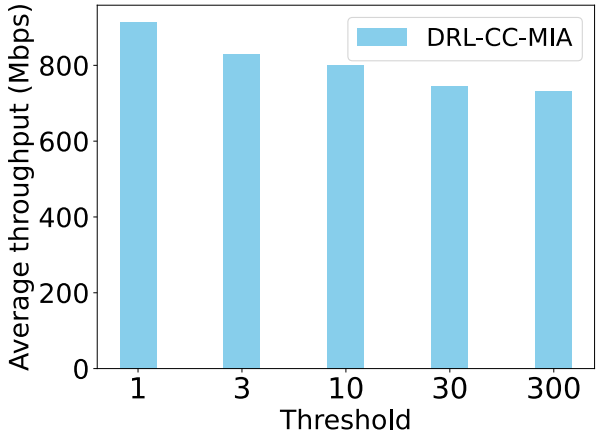
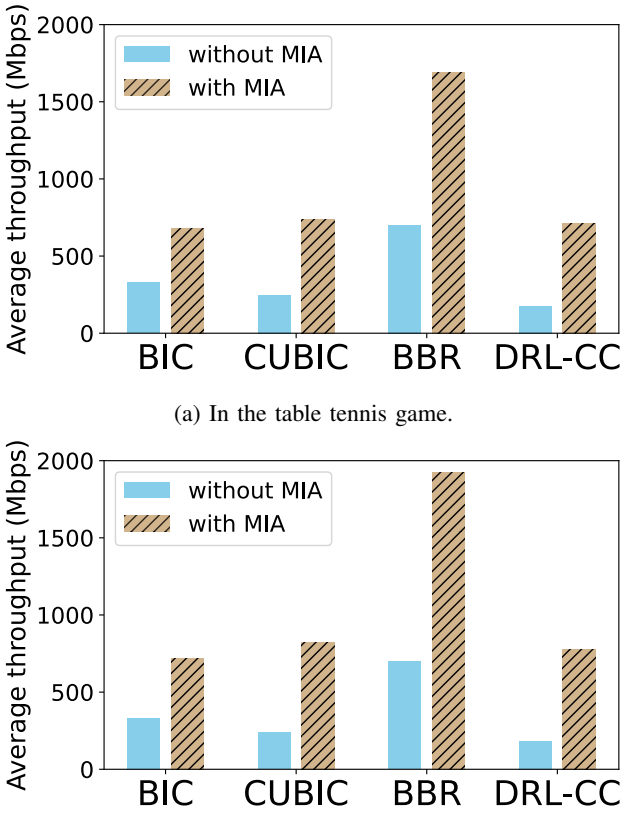


Fig. 10: Average throughput with 10ms average network delay when different MIA threshold numbers are applied to DRL-CC-MIA in the boxing game.

CC-MIA over DRL-CC is substantial without any dispute. As mentioned above, the gains are primarily due to the ability to predict link bandwidth accurately about one RTT ahead into the future, thereby giving TCP senders adequate notice of potential reductions or increases in the data rates offered by the mmWave links and the ability of sending rate adaptations to take advantage of this advanced notice. Finally, the immersive experience is not only sensitive to throughput but also to latency. Fig. 9c shows the average latency for transmitting 1 Gb of data from sender to receiver over the network. Such latencies are important when scenes in the VR game require a large amount of data transmission. To compute this average we randomly sample the traffic flow at 500 different time points and measure the time required to transmitted 1 Gb of data starting at the sample point. Clearly, smaller latencies are better than larger latencies. The figure is a scatterplot of latencies without and with MIA in the four TCP variants for different network delays. Since all the scatter points are substantially below the equal latency line, we can conclude the latencies in the schemes with MIA are considerably smaller as compared to the latency without the MIA.

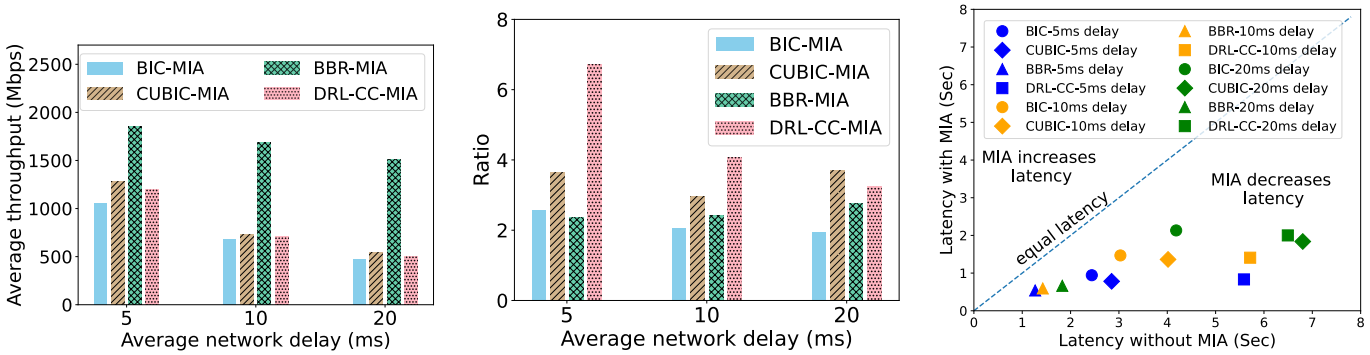


(a) In the table tennis game.

(b) In the beat saber game.

Fig. 11: Average throughput with 10ms average network delay when different TCP variants run with and without MIA.

3) Impact of MIA Threshold (TH_{MIA}): Recall that, the proposed solution uses the following strategy. When the predicted rate is much smaller than the sending rate congestion control scheme, it uses the predicted rate. If the predicted rate is greater than that of the congestion control scheme, it stays with the congestion control scheme if fewer than a pre-specified number of consecutive new acknowledgements have been received. The idea is that this threshold determines when we judge that the network has no hint of congestion and

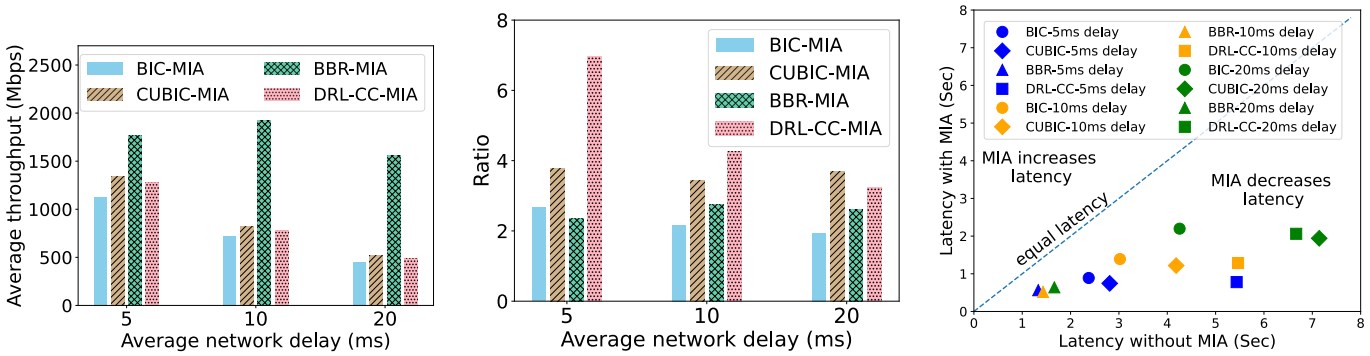


(a) Average throughput when MIA cooperates with different TCP variants.

(b) Improvement ratio when MIA cooperates with different TCP variants.

(c) Data transmission latency comparison between using and not using MIA.

Fig. 12: Performance metrics in the table tennis game when MIA cooperates with different network delays and TCP variants.



(a) Average throughput when MIA cooperates with different TCP variants.

(b) Improvement ratio when MIA cooperates with different TCP variants.

(c) Data transmission latency comparison between using and not using MIA.

Fig. 13: Performance metrics in the beatsaber game when MIA cooperates with different network delays and TCP variants.

it is safe to use the higher data rate of the mmWave link. The larger the value of this threshold the more conservative we are in using a rate higher than that suggested the underlying congestion control scheme. Fig. 10 shows the end-to-end throughput of DRL-CC-MIA as we increase the threshold from 1 to 300. A threshold of 300 means that we use a sending rate higher than that suggested by the congestion control scheme only if we have received at least 300 consecutive new acknowledgements (i.e., not duplicate acknowledgements). Note that, the throughput is not very sensitive to this threshold, although there is a decrease in throughput as the threshold increases. The throughput of DRL-CC-MIA decreases from around 900 Mbps to around 740 Mbps when the MIA threshold increases from 1 to 300.

4) Impact of Games: All the above results are for the game of boxing. Fig. 11, 12, 13 are the same kind of results for two other games, table tennis and beat saber. Fig. 11 is similar to Fig. 8 while Fig. 12 and 13 are similar to Fig. 9. For instance, in Fig. 11a, the average throughput of DRL-CC without MIA is 175 Mbps while the average throughput of DRL-CC with MIA is 710 Mbps during a table tennis game. In Fig. 12 and 13, the throughput improvement with the use of MIA ranges from a factor of 2 to a factor of around 7. Basically, the figures show that the qualitative conclusions for all three of table tennis and beat saber are similar to that for boxing. That, MIA offers significant performance improvement in terms of multi-fold increase in throughput and substantial decrease in latency.

V. Conclusion

To better adapt to the fluctuations in mmWave channel quality, we propose a RL integrated transport-layer Millimeter wave based Immersive Agent (MIA) plugin and provide its implementation in an immersive system architecture. MIA observes the mmWave link bandwidth and the user's speed information during immersive applications in real time and adaptively sends packets based on the prediction with the RL algorithm. The evaluation results show that when the fluctuations in mmWave channel quality impact TCP performance in immersive experience, MIA enhances TCP performance in both end-to-end throughput and latency.

Acknowledgments

This work was partially supported by National Science Foundation grant CNS 1703389, U.S.A and by grant NSTC 108-2221-E-011-058-MY3 and 111-2221-E-011-092 of the National Science and Technology Council, Taiwan, R.O.C.

REFERENCES

[1] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC Editor, RFC 2582, Apr. 1999.

[2] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," *SIGCOMM Computer Communication Review*, vol. 24, no. 4, pp. 24–35, Oct. 1994.

[3] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *Proceedings of International Conference on Computer Communications (INFOCOM)*, vol. 4, Nov. 2004, pp. 2514–2524.

[4] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-friendly High-speed TCP Variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[5] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for High-speed and Long Distance Networks," in *Proceedings of International Conference on Computer Communications (INFOCOM)*, Apr. 2006, pp. 1–12.

[6] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *ACM Queue*, vol. 14, pp. 20–53, Sep. 2016.

[7] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting Congestion Control for Consistent High Performance," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, May 2015, pp. 395–408.

[8] H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks," *ACM Wireless Networks*, vol. 1, no. 4, pp. 469–481, Dec. 1995.

[9] H. Haile, K.-J. Grinnemo, S. Ferlin, P. Hurtig, and A. Brunstrom, "End-to-end congestion control approaches for high throughput and low delay in 4G/5G cellular networks," *Computer Networks*, vol. 186, p. 107692, Feb. 2021.

[10] P. Popovski, Č. Stefanović, J. J. Nielsen, E. de Carvalho, M. Angjelichinoski, K. F. Trillingsgaard, and A. Bana, "Wireless Access in Ultra-Reliable Low-Latency Communication (URLLC)," *IEEE Transactions on Communications*, vol. 67, no. 8, pp. 5783–5801, Aug. 2019.

[11] S. K. Saha, S. Aggarwal, R. Pathak, D. Koutsonikolas, and J. Widmer, "MuSher: An Agile Multipath-TCP Scheduler for Dual-Band 802.11ad/ac Wireless LANs," in *The 25th Annual International Conference on Mobile Computing and Networking*, Oct. 2019, pp. 1–16.

[12] Z. Wu, C.-Y. Huang, and P. Ramanathan, "COded Taking And Giving (COTAG): Enhancing Transport Layer Performance over Indoor Millimeter Wave Access Networks," in *ICC 2022 - IEEE International Conference on Communications*, 2022, pp. 3610–3616.

[13] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Wanna Make Your TCP Scheme Great for Cellular Networks? Let Machines Do It for You!" *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 265–279, Jan. 2021.

[14] Z. Xu, J. Tang, C. Yin, Y. Wang, and G. Xue, "Experience-Driven Congestion Control: When Multi-Path TCP Meets Deep Reinforcement Learning," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1325–1336, Jun. 2019.

[15] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, "Owl: Congestion Control with Partially Invisible Networks via Reinforcement Learning," in *Proceedings of International Conference on Computer Communications (INFOCOM)*, Jul. 2021, pp. 1–10.

[16] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[17] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR*, May 2016.

[18] K. Winstein and H. Balakrishnan, "TCP Ex Machina: Computer-Generated Congestion Control," in *Proceedings of the ACM SIGCOMM*. Association for Computing Machinery, Aug. 2013, pp. 123–134.

[19] P. Geurts, I. El Khayat, and G. Leduc, "A machine learning approach to improve congestion control over wireless computer networks," in *Fourth IEEE International Conference on Data Mining (ICDM'04)*, Nov. 2004, pp. 383–386.

[20] G. Sun, C. Li, Y. Ma, S. Li, and J. Qiu, "End-to-End TCP Congestion Control as a Classification Problem," *IEEE Transactions on Reliability*, pp. 1–11, May 2022.

[21] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic Meets Modern: A Pragmatic Learning-Based Congestion Control for the Internet," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2020, p. 632–647.

[22] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[23] S. Sur, I. Pefkianakis, X. Zhang, and K.-H. Kim, "WiFi-Assisted 60 GHz Wireless Networks," in *Proceedings of the Annual International Conference on Mobile Computing and Networking*, Oct. 2017, pp. 28–41.

[24] Z. Wu, C.-Y. Huang, and P. Ramanathan, "Measuring Millimeter Wave Based Link Bandwidth Fluctuations During Indoor Immersive Experience," *IEEE Networking Letters*, vol. 4, no. 3, pp. 113–117, 2022.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR*, May 2015.

[26] C. Dovrolis, P. Ramanathan, and D. Moore, "Packet Dispersion Techniques and Capacity Estimation," *IEEE/ACM Transactions on Networking*, vol. 12, no. 6, pp. 963–977, Dec. 2004.

[27] Y. Ghasempour, C. R. C. M. Da Silva, C. Cordeiro, and E. W. Knightly, "IEEE 802.11ay: Next-Generation 60 GHz Communication for 100 Gb/s Wi-Fi," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 186–192, Dec. 2017.

[28] G. F. Riley and T. R. Henderson, *The ns-3 Network Simulator*. Springer Berlin Heidelberg, 2010, pp. 15–34.

[29] P. Gawłowicz and A. Zubow, "ns-3 meets OpenAI Gym: The Playground for Machine Learning in Networking Research," in *ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, Nov. 2019.

[30] G. Brockman *et al.*, "OpenAI Gym," *CoRR*, vol. abs/1606.01540, Jun. 2016. [Online]. Available: http://arxiv.org/abs/1606.01540