BoB: Bandwidth Prediction for Real-Time Communications Using Heuristic and Reinforcement Learning

Abdelhak Bentaleb, *Member, IEEE,* Mehmet N. Akcay, May Lim, Ali C. Begen, *Senior Member, IEEE,* and Roger Zimmermann, *Senior Member, IEEE*

***Abstract*—Bandwidth prediction is critical in any Real-time Communication (RTC) service or application. This component decides how much media data can be sent in real time. Subsequently, the video and audio encoder dynamically adapts the bitrate to achieve the best quality without congesting the network and causing packets to be lost or delayed. To date, several RTC services have deployed the heuristic-based Google Congestion Control (GCC), which performs well under certain circumstances and falls short in some others. In this paper, we leverage the advancements in reinforcement learning and propose BoB (Bang-on-Bandwidth) — a hybrid bandwidth predictor for RTC. At the beginning of the RTC session, BoB uses a heuristic-based approach. It then switches to a learning-based approach. BoB predicts the available bandwidth accurately and improves bandwidth utilization under diverse network conditions compared to the two winning solutions of the ACM MMSys'21 grand challenge on bandwidth estimation in RTC. An open-source implementation of BoB is publicly available for further testing and research.**

***Index Terms*—Bandwidth prediction; real-time communications; reinforcement learning; RTC; WebRTC; AlphaRTC.**

I. Introduction

Real-time Communication (RTC) services account for a sizeable fraction of today's Internet traffic [23]. For example, there were 300 million daily meeting participants on the Zoom platform alone in 2020, a 50% increase from 2019 [65], and on the Facebook Messenger application, there were 150 million daily video calls in 2021 [49]. With ubiquitous connectivity and more efficient video and audio codecs, RTC services continue to grow and evolve.

Today, RTC is used in a range of applications such as video gaming [53], [43], [26], videoconferencing [38], [34], e-learning [9] and real-time immersive experience sharing [59]. Needless to say, RTC is an integral part of our lives as it enables us to stay connected with the rest of the world while working remotely, which has become the new normal due to the COVID-19 pandemic. However, this does not mean users' quality of experience (QoE) for RTC services is always great. Occasionally and sometimes more than occasionally, users still suffer from blurry, low-quality or distorted video, high latency or video freezes and audio drops.

To date, there has been significant research to improve QoE in RTC services. These efforts offered several solutions that can be divided into three broad categories: (i) congestion control optimization at the transport layer [22], [24], [60], [64] that primarily aims to provide an accurate bandwidth estimation, (ii) bitrate selection optimization for video codecs [68] (*e.g.*, H.26x, VPx and AV1) that strives to adapt the bitrate (through the rate control at the application layer) for each frame to suit the instantaneous network capacity changes, and (iii) mixed techniques that combine congestion control and bitrate selection optimizations. Despite the advances in codec rate control, accurate bandwidth estimation is still an open problem. It plays a critical role in maintaining a good QoE as the codec allocates more or fewer bits based on this estimation. In other words, if the actual bandwidth is overestimated or underestimated, this can be detrimental to the QoE. Existing heuristics (*e.g.*, [22], [21]) may work well in some network environments but not so well in others [27] due to dynamic, complex and diverse bandwidth fluctuations. These heuristics mainly follow the Google Congestion Control (GCC) algorithm [1] that implements two rules that consider the aggregated Real-time Transport Protocol (RTP, RFC 3550) feedback information to estimate the bandwidth. The first rule is a loss-based rate controller implemented at the sender, while the second is a delay-based one implemented at the receiver.

Deep reinforcement learning (DRL) has recently emerged as a key solution for many networking problems such as bitrate adaptation [45], congestion control in TCP [7], [42] and RTC [27], scheduling [44] and bandwidth prediction [13]. Leveraging the power of a learning-based approach that masters and adapts dynamically to various environments, we design BoB (Bang-on-Bandwidth) — a bandwidth predictor for RTC. BoB is located at the receiver and operates fully automatically by learning from experience and reacting quickly to changes in network conditions while considering video quality and packet delay/loss. It uses actor-critic networks for model training and Proximal Policy Optimization (PPO) [52] with clip and Adam optimizers for policy updates at each time interval. Using DRL directly in the context of bandwidth prediction requires a certain level of caution because of the cold start issues (*i.e.*, not enough data being available at the beginning of the session) [66]. The reason is that DRL approaches are often trained offline with large

A. Bentaleb is with Gina Cody School of Engineering and Computer Science, Concordia University, Canada (e-mail: abdelhak.bentaleb@concordia.ca)

M. Lim and R. Zimmermann are with the School of Computing, National University of Singapore, Singapore (e-mail: {maylim,rogerz}@comp.nus.edu.sg).

Mehmet N. Akcay and Ali C. Begen are with Ozyegin University, Istanbul, Turkey (e-mail: necmettin.akcay@ozu.edu.tr; ali.begen@ozyegin.edu.tr).

amounts of data, and then used online with limited data. Such a gap between offline and online environments results in inconsistent performance [66] caused by taking sub-optimal actions. To avoid this issue, BoB includes an adaptive selector for bandwidth prediction that initially uses a heuristic-based controller. Once it collects sufficient input data, it switches to a learning-based controller.

Note that we may use estimation and prediction interchangeably throughout this paper while keeping a small but important difference adopted from [13]. An estimation is derived from the raw measurements and/or samples using simple *smoothing* techniques, whereas a prediction is derived from the smoothed values and/or other data using learning-based techniques.

The contributions of this paper are three-fold:

1) We design BoB, a receiver-side hybrid bandwidth prediction solution for RTC, which combines a heuristic-based controller (inspired by the GCC algorithm) with a DRL controller. The main feature of BoB is to leverage the DRL benefits in adapting to diverse network conditions while using the heuristic-based controller only at the beginning of an RTC session when input data is scarce.
2) We propose an adaptive technique to select between the heuristic and learning-based controllers to avoid inaccurate actions when using DRL for bandwidth prediction.
3) We implement BoB on Microsoft's OpenNetLab platform termed AlphaRTC [5] and validate its performance gains against the recent state-of-the-art solutions and winners of the grand challenge organized by Microsoft and OpenNetLab on the subject of bandwidth estimation for RTC at ACM MMSys 2021 [3]. To train BoB's DRL model, we incorporate BoB into RTC GYM [3], which emulates an RTC environment, and subsequently, use the model for evaluation using real-world network traces (online BoB model inference). We also evaluate BoB in the wild using the OpenNetLab public Internet-based testbed. Evaluation results show that BoB achieves good prediction accuracy with high utilization and viewer experience across many real-world network conditions. The source code for BoB is publicly available at [11].

The rest of the paper is organized as follows: Section II overviews some of the QoE optimization solutions for RTC systems. Section III details the proposed learning-based solution (BoB) for bandwidth prediction in RTC systems. The evaluation and analysis are given in Section IV, followed by a discussion on open directions in Section V. Finally, Section VI concludes the paper.

II. Related Work

Improving QoE for different video streaming services, such as RTC, has gained massive attention in the last several years. For this purpose, solutions have been developed with techniques ranging from heuristics to learning-based methods at the transport layer (*e.g.*, congestion control) and application layer (*e.g.*, bitrate selection and bandwidth estimation). In general, these solutions fall into three main categories.

A. Congestion Control Optimization

There are many congestion control solutions that include numerous variants of TCP. Here, we briefly present some of them. Among the early solutions, TCP Reno [37] and NewReno [29] both use a heuristic additive-increase-multiplicative-decrease (AIMD)-based algorithm that considers packet loss as the key indicator for congestion. Later, improved congestion control versions have emerged, such as TCP Cubic [31] and TCP Vegas [18] (and then Copa [8]), where the former tries to replace the AIMD function with an improved one while the latter uses delay as the primary indicator of congestion instead of packet loss. More recently, BBR [19] uses delay instead of loss as the primary parameter to determine the sending rate, allowing it to work near the optimal point of full bandwidth utilization and low delay. BBRv2 [20] aims to address the issues that were introduced in the initial version: (i) unfairness, and (ii) excessive retransmissions in shallow-buffered networks.

As learning techniques became popular, there were attempts to automatically perform the task of congestion control. Winstein *et al.* [58] designed Remy, a distributed congestion control solution for heterogeneous and dynamic network environments. Remy formulates congestion control as an optimization problem and implements an offline mapping from all possible events to good actions using a dynamic programming approach. Using online learning techniques, PCC-Vivace [25] was proposed to select the best sending rates automatically. Indigo [62] adjusts the congestion window based on a trained model that employs imitation-learning, while Aurora [39] leverages basic DRL techniques to determine the sending rate. Orca [7] uses a hybrid approach that combines a legacy congestion control solution with modern DRL techniques. Zhu *et al.* [70] proposed NADA, a congestion control scheme for interactive RTC services, where the sender adjusts its sending rate based on either implicit or explicit congestion signaling markings from the network nodes. Johansson *et al.* designed SCReAM [41], a hybrid loss and delay-based congestion control algorithm for interactive video streaming applications. Interested readers are encouraged to read more details in [62], [48].

B. Bitrate Selection Optimization

Fang *et al.* [27] designed an RL-based agent to control the sending rate in an RTC system. Their preliminary results showed good performance under challenging network conditions. Tianrun *et al.* [55] designed Gemini, an ensemble framework for bandwidth estimates in RTC. Gemini implements a hybrid technique that switches between the heuristic-based GCC rule and a DRL agent on the fly based on a safety factor. This safety factor decides when Gemini falls back to the GCC rule once the DRL model performs poorly and then switches back to DRL when the performance improves. However, based on our experimental test runs and results (Section IV), Gemini suffers from three issues: (i) the switching technique fails frequently when it tries to select the correct algorithm, especially under challenging network conditions with a high packet loss ratio (*e.g.*, 3G/4G), (ii) the DRL algorithm uses a simple neural network that does not consider the fluctuation in the past bandwidth prediction values, and (iii) the DRL algorithm fails to converge to perform the best bandwidth prediction decisions. Such issues may result in bandwidth overpredictions or underpredictions.

Similarly, Wang *et al.* [57] proposed HRCC, which uses an RL agent to dynamically tune the values of GCC parameters depending on the network variability instead of using fixed values to boost bandwidth estimation accuracy. Our solution (BoB) falls into this category and its objective of controlling the receiving rate is similar to HRCC and Gemini. All these solutions (HRCC, Gemini and BoB) use a GCC-like heuristic algorithm. However, the key differences are in the DRL-agent design. BoB differs from Gemini in the following aspects: (i) the DRL architecture and set of the NN inputs, (ii) the adaptive algorithm switcher, where during the streaming session, Gemini keeps switching between the DRL and heuristic algorithms, and BoB only uses the heuristic at the beginning and then switches to DRL once more data is available, and (iii) Gemini uses an ACK-based heuristic algorithm while BoB uses a delay-loss based heuristic algorithm.

There exist other solutions for improving QoE in RTC, such as rate adaptation and control like QARC [35] and Proteus [60], hybrid error correction [32], and multipath [67]. In HTTP adaptive streaming, there exist multiple studies on bitrate adaptation [16] that leverage different approaches, such as software-defined networking [14], network-assisted streaming [17], client-driven optimization [36], [54], control theory [63], queuing theory [61], machine learning [45], [33], [15], scheduling [40] and game theory [12].

C. Mixed Techniques

Fouladi *et al.* [30] designed Salsify as an RTC architecture that includes a video codec and a network transport protocol. Salsify uses per-frame rate adaptation and aims to work under extreme network conditions by alleviating packet losses and delays. To achieve this, Salsify employs a custom encoding/decoding scheme not supported by existing hardware codecs. Zhang *et al.* [67] proposed a solution that combines a multipath transmission scheme with path selection for improved transmissions in RTC. Here, the sender selects the best path from several candidate paths using a multi-armed bandit learning-based technique. Zhou *et al.* [69] proposed Concerto, a machine learning-based bitrate adaptation system aiming to maximize video telephony QoE. Concerto first extracts high-level features of both layers (application and transport) and then leverages deep imitation learning to train models using massive data traces. In particular, it considers historical packet losses, packet delays and the sending/receiving rates in its neural network and imitates the behavior of an expert (an Oracle that knows the actual bandwidth values). Zhang *et al.* [66] developed an online RL-based solution for rate decisions in RTC systems named OnRL. The central insight behind OnRL is that RL models trained offline in a simulator suffer from less satisfactory performance when deployed under real conditions.

III. BoB: Bang-on-Bandwidth

Predicting the bandwidth is one of the critical tasks in RTC that directly impact the user experience. The essential question is how to perform bandwidth prediction accurately, considering the collected information from the Real-time Transport Protocol (RTP, RFC 3550) packets. Information that includes sending/receiving time and packet size can be collected with every received RTP packet. This information is used to compute the receiving rate, packet delay and packet loss, all used as input to figure out how much available bandwidth there is now and will be soon on the current network path. Typically, bandwidth prediction is performed using a heuristic-based scheme (*e.g.*, GCC-based [1]). In a learning-based approach, the above inputs are translated into a state and reward (QoE), which are then mapped to an action (bandwidth prediction). BoB achieves the benefits of both approaches to perform the bandwidth prediction task as explained below.

A. Overview

The overall workflow of BoB is depicted in Fig. 1. It consists of two phases: BoB testing and BoB training.

1) BoB Training Phase: We use the AlphaRTC GYM simulator [5], based on an ns-3 [51] and WebRTC implementation. This simulator emulates a WebRTC session, utilizing various network traces that were collected from real-world environments such as Belgium 4G/LTE [56], Norway 3G/HSDPA [50] and NYU LTE [46]. Each network trace is comprised of a throughput value, a round-trip time (RTT) and a packet loss ratio. We implemented BoB as a bandwidth prediction module within RTC GYM. During the WebRTC session, the simulator collects and computes statistics (*e.g.*, receiving rate, packet delay and loss) from each received RTP packet, and then these statistics are fed as inputs into BoB, which in turn predicts the bandwidth at every time step. The predicted bandwidth is sent to the sender via RTP Control Protocol (RTCP, RFC 3550) feedback to adjust the encoding rate. During the offline phase, we train our learning-based BoB model (refer to the DRL controller in Fig. 2, with more details given in Section III-C2) and this model will be used during the testing phase. We note that because our experiments were conducted using a short video sample, we did not retrain the BoB model during the testing phase. However, if/when desired, it could be retrained periodically.

2) BoB Testing Phase: We use the AlphaRTC implementation [5] with the BoB controller for a receiver-side hybrid bandwidth predictor, as highlighted in blue color in Fig. 1. The system consists of an (RTP) sender and (RTP) receiver. The sender initiates the RTC video session with the receiver by creating a UDP socket to send RTP packets and receive RTCP feedback. The congestion control is adapted from GCC and includes two controllers: a loss-based controller and the hybrid BoB controller (it is called a hybrid as it has both a heuristic delay-based controller and a learning-based DRL controller). The BoB controller is placed at the receiver and responsible for computing a bitrate (x^r) based on the BoB bandwidth predictor output, which is then fed back to the sender. Conversely, the

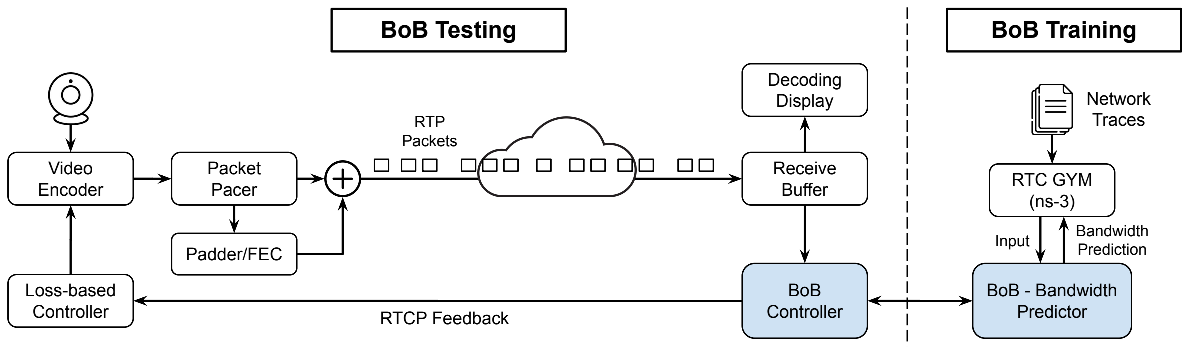


Fig. 1: Overall workflow of BoB.

loss-based controller is placed at the sender and is responsible for computing the target sending rate (denoted by x^s). The target bitrate x^s is fed to the video encoder, which attempts to encode the video at a bitrate as close to the target as possible. The encoded video is then forwarded to the packet pacer responsible for regulating the bitrate produced by the encoder when the bitrate of the encoded video deviates from the target. Here, the encoder cannot change the rate as frequently as the pacer rate. If the video encoder produces a bitrate higher than the target, then the pacer is allowed to drain its queue at a higher rate to alleviate queuing delays at the sender. On the other hand, padding/forward error correction (FEC) can be added, if desired, under certain circumstances. This way, on average, the sending rate is expected to be equal to the target bitrate x^s.

B. System Architecture

BoB is a hybrid rate control solution implemented at the receiver to improve the QoE for RTC systems. It combines the strength of a heuristic-based rate controller with a DRL-based controller to predict the bandwidth. As shown in Fig. 1, BoB takes the historical packet-level statistics from the network path as an input, where we denote the receiving rate by c_t, packet delay intervals by d_t, packet loss ratio by l_t and the n most recent predicted bandwidth samples by $\overrightarrow{X_t^r} = \{x_{t-1}^r, x_{t-2}^r, \ldots, x_{t-n}^r\}$). It outputs a prediction (denoted by $a_t^\star = x_t^r$) for the next t-th time window denoted by W_t (in milliseconds), where $t = \{1, 2, \ldots, T\}$ and T is the total number of time windows of an RTC session. The predicted bandwidth value is then sent to the sender using an RTCP feedback message, which in turn is passed to the encoder. After that, the encoder uses this value as the target bitrate and encodes the frames based on this target. Therefore, BoB controls the receiving rate and helps to avoid issues that could lead to poor QoE. In short, BoB replaces the traditional, heuristic-only-based rate controller (*e.g.*, based on an unscented Kalman filter) by leveraging the power of DRL. During the offline training phase, it uses the past and current information of the incoming packets (at the transport layer) as input to the neural network. Due to the nature of DRL, BoB might deviate from the right decision in some corner (uncovered) cases, which mostly happen at the beginning of an RTC session. For this reason, we developed a simple but robust adaptive selector that enables run-time switching between the

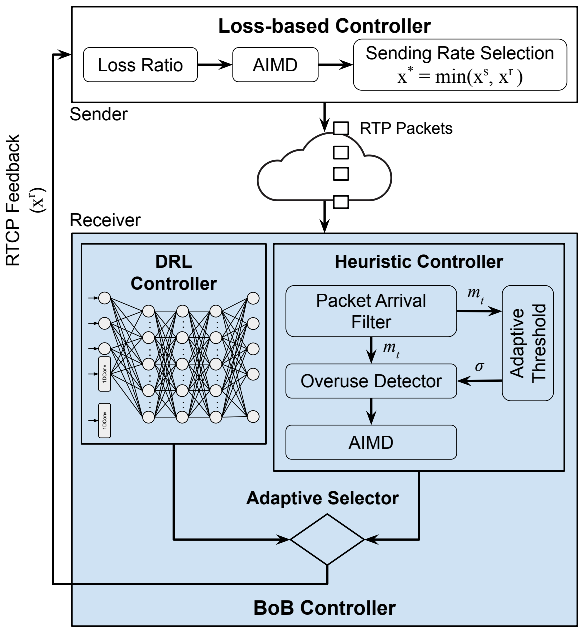


Fig. 2: Receiver-side BoB controller (blue) and sender-side loss-based controller.

heuristic and DRL-based controllers. The adaptive selector uses the heuristic-based controller at the beginning of an RTC session when the DRL controller behaves sub-optimally because of limited session data and the incorrect exploitation actions. Then, it switches to the DRL controller once most of the corner cases are covered and the predicted values become accurate. Specifically, it uses a current percentage value and a percentage threshold (*i.e.*, fixed to 30%) that is tuned empirically as a switching point between the heuristic-based and DRL controllers. The current percentage value is computed based on the difference and average in the predicted bandwidth values given by the heuristic and DRL controllers.

As shown in Fig. 2, each endpoint (sender or receiver) runs its controller. The receiver runs the BoB controller, whereas the sender runs a loss-based controller. Next, we describe the receiver-side BoB controller and the sender-side loss-based controller in detail.

C. (Receiver-Side) BoB Controller

Here, we describe the BoB controller, which consists of (i) a delay-based (heuristic) controller, (ii) a DRL controller, and (*iii*) an adaptive selector.

1) Delay-Based (Heuristic) Rate Controller: At each time window W_t, the delay-based rate controller predicts the bandwidth x^r_t as described in Algorithm 1. In this algorithm, β = 1.08 and α = 0.85 are coefficients of the packet arrival Kalman filter, which are tuned empirically based on our experiments, σ is the controller's state, c_t is the receiving rate measured in the last $W_t = 200$ milliseconds (ms), and $\bar{x}_t$ is the additive value that is determined by the rate control *region*. The delay-based controller first uses the packet arrival filter that divides and groups the received packets into 200-ms windows and then computes the slope factor (denoted by m_t) based on a delay gradient between the groups of received packets and judges the trend of the delay change. After that, m_t is fed to the adaptive threshold, which sets the threshold used by the overuse detector. Then, the overuse detector produces a signal that drives the network state (denoted by τ): *underuse*, *overuse* or *normal* based on m_t and threshold (see Fig. 2). The network state is then mapped to a controller state *increase*, *decrease* or *hold* using an AIMD algorithm to predict the currently available bandwidth according to the prevailing network state. If the controller state is *decrease*, then the controller sets the rate control *region* to state NearMax. Once the controller state is changed to *increase* and the rate control *region* is in state NearMax, the controller sets $\bar{x}_t = c_t$. Otherwise, if the controller state is *increase* and the rate control *region* is in state of MaxUnknown, the controller sets $\bar{x}_t = \beta \times c_t$. Therefore, the controller additively increases x^r_t based on the rate control *region*.

Algorithm 1 Delay-based Rate Controller

```
1:  function HEURISTICCONTROLLER($c_t, d_t, l_t, X^r_t$)
2:      $\alpha \leftarrow 0.85$, $\beta \leftarrow 1.08$, region ← MaxUnkown
3:      for Each time window $W_t$, $W_t > 0$ do                ▷ every 200 ms
4:          $\sigma \leftarrow$ GetControllerState()               ▷ overuse detector
5:          $\hat{c}_t \leftarrow$ std($C_t$)                      ▷ standard deviation of $C_t$
6:          $\bar{c}_t \leftarrow$ Average($C_t$)          ▷ $C_t = \{c_t, c_{t-1}, \dots, c_{t-n-1}\}$
7:          switch $\sigma$ do
8:              case 'Increase'
9:                  if ($c_t > \bar{c}_t + 3 \times \hat{c}_t$) then
10:                     $\tau \leftarrow$ 'Underuse'               ▷ $\tau$: network state
11:                     region ← MaxUnkown
12:                 end if
13:                 if region == MaxUnkown then
14:                     $\bar{x}_t \leftarrow \beta \times c_t$
15:                     $x^r_t \leftarrow x^r_{t-1} + \bar{x}_t$
16:                 else if region == NearMax then
17:                     $\bar{x}_t \leftarrow c_t$
18:                     $x^r_t \leftarrow x^r_{t-1} + \bar{x}_t$
19:                 end if
20:             case 'Decrease'
21:                 $x^r_t \leftarrow \alpha \times c_t$
22:                 $x^r_t \leftarrow$ Min($x^r_t$,$\alpha \times x^r_{t-1}$)
23:                 region ← NearMax
24:             case 'Hold'
25:                 $x^r_t \leftarrow x^r_{t-1}$
26:         $\tau \leftarrow$ 'Underuse'
27:         Return($x^r_t$)
28:     end for
29: end function
```

2) Learning-Based (DRL) Rate Controller: BoB implements an RL agent that interacts with the *environment* encompassing the communication process between the sender and receiver in the RTC system. For the BoB model training, the packet-level statistics (input) are collected periodically during a fixed time window of $W_t = 200$ ms and aggregated as the environment state. Subsequently, the agent predicts the bandwidth that represents an *action* value. Formally, the RL agent interacts with the environment that defines a state space denoted by $\mathcal{S}$. At each time window W_t (at time epoch t), the RL agent receives a state $s_t \in \mathcal{S}$ from the environment and then takes an action $a_t \in \mathcal{A}$ (bandwidth prediction for the next time window W_{t+1}) while it receives a reward $r_t \in \mathcal{R}$. The essential objective of the agent is to find an optimal policy $\pi^\star : \mathcal{S} \rightarrow \mathcal{A}$ that maps states-to-actions, maximizing the overall reward (*i.e.*, finding the bandwidth that maximizes the receiving rate while minimizing the packet loss and delay). After the bandwidth prediction action a_t is taken, the BoB environment observes the new receiving rate, packet loss, delay and the predicted bandwidth, and transits to the next state $s_{t+1} \in \mathcal{S}$, while updating the reward $r_{t+1} \in \mathcal{R}$. The DRL controller is depicted in Fig. 3.

a) ***Input State Space and Network:*** At each time window W_t, the state input is a 1×11 vector of 11 dimensions defined as $s_t = \{c_t, d_t, l_t, \overrightarrow{X^r_t}\}$, comprised of the receiving rate c_t (bps), packet delay[1] d_t (ms), packet loss ratio (%) and the n most recent bandwidth prediction samples $\overrightarrow{X^r_t}$ (bps). We normalize each state input using a `linear-to-log()` function in a value-range [0,1]. We then feed the current state s_t as the input to the actor-critic network that comprises two neural networks. As depicted in Fig. 4, $\overrightarrow{X^r_t}$ is fed into a 1DConv (LSTM) layer in time order for feature extraction. The main insight behind using LSTM is capturing the bandwidth variation's temporal characteristics. Thus, the accuracy of the bandwidth prediction can be improved. Other inputs are fed into a linear, fully-connected (FC) layer with a Rectified Linear Unit (`ReLU()`) activation function. After that, the input layers are concatenated and finally fed into the hidden layers. Results from the concatenation are then aggregated in three levels of FC layers that use 514, 320 and 64 neurons, with a `ReLU()` activation function with a slope of 0.5.

We use the same structure for both actor and critic networks, but with different outputs. For the actor network, we use a `softmax()` distribution function followed by a logarithm (`log_softmax()`) as the last FC layer with L2 normalization of the network, resulting in an output in the range from 0 to 1. The output (selected action) is then mapped to a value between 0.01 to 8 Mbps (as fixed in AlphaRTC [5], [27]) as the bandwidth prediction using a `log`-to-`linear()` function. The critic network is similar to the actor without `log_softmax()` in the last layer, resulting in output state-values, denoted by $V^{\pi_\theta}(s_t, w)$ (value function), that help the actor network update the policy distribution in the direction suggested by the critic network (such as with policy gradients). We note that each 1DConv layer uses a 3×3 convolution with 64 filters to extract implicit features, and is followed by a `ReLU()` activation function that tries to maintain a non-zero policy gradient over the whole training phase. Therefore, the

[1]The packet delay is the average RTT.

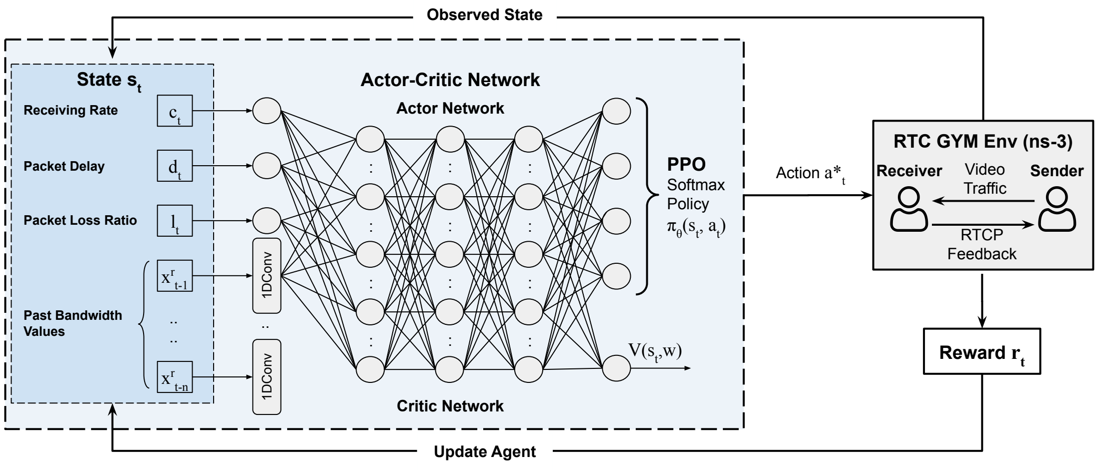


Fig. 3: Learning-based (DRL) rate controller for BoB.

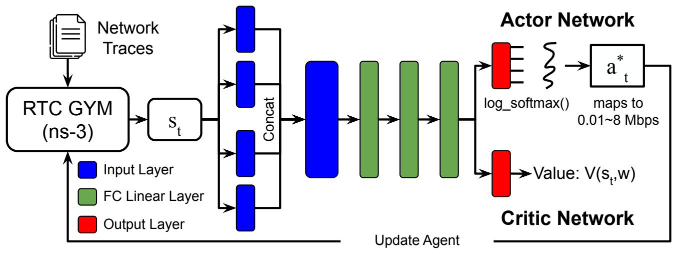


Fig. 4: The neural network design for training in BoB.

vanishing gradient problem is avoided while the training time is reduced.

b) ***Action Space:*** In each time window W_t, BoB policy $\pi^\star_\theta$ maps s_t to a compact action space whose values range between 0.01 and 8 Mbps. Specifically, $\mathcal{A} = \{a_0 : 0.01 - 2$ Mbps, $a_1 : 2-4$ Mbps, $a_2 : 4-6$ Mbps, $a_3 : 6-8$ Mbps$\}$, representing an appropriate range of bandwidth prediction for RTC systems. Therefore, the output is a 1×4-dimensional vector that identifies the state-action probabilities produced by `log_softmax()`. Then, $\pi^\star_\theta : s_t \rightarrow a^\star_t$ maps the state to a suitable action ($a^\star$ = $[a_0$,$a_3]$) based on the state-action probabilities, *i.e.*, the agent policy selects the action with the highest probability.

c) ***Reward Function:*** The reward r_t is calculated after each action a_t is taken to ensure that BoB can learn from past experience. It reflects the performance of the bandwidth prediction accuracy according to the user QoE. At each time window W_t, we define r_t based on [27] as follows:

$$r_t = \texttt{linear-to-log}(c_t) - \min(d_t/1000, 1) - l_t.$$

The agent is rewarded when it receives more packets (leading to higher QoE) and penalized when packet delay/loss increases (leading to lower QoE).

d) ***Training Algorithm:*** We use the Advantage Actor-Critic with on-policy Proximal Policy Optimization (PPO) and the Adam optimizer for policy updates. During the training, the objective of BoB is to maximize the total discounted cumulative reward, which is expressed as:

$$R_t = \sum_{\bar{t}=t}^{T_{\pi_\theta}} \gamma^{\bar{t}-t} \times r_t,$$

where T_{π_θ} denotes the batch size for updating the gradient policy (fixed to 4,000 time windows per episode in our simulations), $\gamma \in [0, 1]$ serves as a discount factor (usually customized as 0.99 or 0.9) and R_t represents the discounted cumulative reward from time t to the end of the RTC session. The objective of the actor network is to find a policy π : $\pi^\star_\theta(s, a) \rightarrow [0, 1]$ to maximize R_t, where $\pi^\star_\theta : s \rightarrow a^\star$ is the probability distribution over different actions $\mathcal{A}$. The stochastic policy $\pi^\star_\theta$ is responsible for selecting an action $a^\star$ with the highest probability. On the other hand, the critic network is responsible for making an objective assessment for each current state s_t using a value function $V^{\pi_\theta}(s_t, w)$.

In the training algorithm, we use PPO and the Adam optimizer to update the gradient policy such that R_t is maximized at every training episode as:

$$\triangledown \bar{R}_t = \frac{1}{\Theta} \sum_{\theta=1}^{\Theta} \sum_{t=1}^{T_\theta} A^{\pi_\theta}_t(s_t, a_t) \triangledown \log \pi_\theta(a_t, s_t),$$

where Θ is the total number of episodes, $A^{\pi_\theta}(s_t, a_t)$ (= $R_t - b_t$) is the advantage function that expresses the difference in the cumulative reward between the actual value after selecting the action a_t based on policy π_θ at s_t and the expected value. The advantage function is calculated as a function of R_t and baseline b_t that has a significant impact on the convergence of the total cumulative reward R_t. In the DRL model, we found A^{π_θ} did not work well. Hence, we replaced it with $A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t, w)$. Q^{π_θ} is computed by the actor network, which uses the k-step Temporal Difference (TD) method given by:

$$Q^{\pi_\theta}(s_t, a_t) = \sum_{k=0}^{k=\Theta-1} \gamma^k r_{t+k} + \gamma^\Theta V(s_{t+\Theta}, w).$$

For each training step, the actor network strives to maximize R_t through maximizing A^{π_θ}, *i.e.*, making better action decisions than the current policy π. Therefore, the parameter θ of the actor is updated via a stochastic gradient ascent algorithm as follows:

$$\theta \leftarrow \theta + \alpha \sum_{t=1}^{T_\theta} A^{\pi_\theta}_t(s_t, a_t) \triangledown_\theta \log \pi_\theta(a_t, s_t),$$

where α is the learning rate and $\bigtriangledown_\theta \log \pi_\theta(a_t, s_t)$ represents the dynamics that parameter θ accounts for in order to achieve the objective. It is worth noting that BoB leverages dropouts with probability ($p = 0.5$) to add a regularization term to the update of the actor network, which helps to alleviate overfitting issues. Such a regularization term can be considered the entropy of the probabilities over the bandwidth prediction decisions $H(\pi_\theta(.|s_t))$, which promotes exploration and avoids severe overfitting.

The critic network is responsible for making an objective assessment for all the states $\forall s_t \in \mathcal{S}$ during the training. To do so, the critic network uses the standard TD method to compute the loss function and minimize its value. Hence, the parameter w of the critic network is updated through a stochastic gradient descent algorithm as follows:

$$w \leftarrow w - \alpha \sum_{t=1}^{T_w} \bigtriangledown_\theta (r_t + \gamma V^{\pi_\theta}(s_{t+1}, w) - V^{\pi_\theta}(s_t, w))^2,$$

where $V^{\pi_\theta}(s_t, w)$ and $V^{\pi_\theta}(s_{t+1}, w)$ are the objective assessments for s_t and s_{t+1}, respectively, from the critic network.

We update the policy π_θ periodically every k steps $< T_\theta$ (update interval) using PPO with clipped objective and the Adam optimizer. The PPO aims to optimize (via Adam) the following clipped objective function:

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = E\Big[\sum_{t=0}^{T_\theta}[\min(ratio_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t^{\pi_k}]\Big]$$

$$\theta_{k+1} = \arg\max_\theta \mathcal{L}_{\theta_k}^{CLIP}(\theta),$$

where E denotes the empirical expectation over time steps, $ratio_t(\theta)$ (= $\pi_\theta(s_t, a_t)$/ $\pi_{\theta_{old}}(s_t, a_t)$) is the ratio of the probabilities under the new and old policies, and ε is the clip hyperparameter (usually fixed to 0.1 or 0.2).

3) Adaptive Selector: The main purpose of the adaptive selector is to decide when to switch between the heuristic and learning-based rate controllers. With this functionality, we enable a hybrid bandwidth prediction and increase the accuracy of the DRL controller in the long term. Bandwidth prediction is likely to be inaccurate (because of bandwidth underprediction caused by the lack of data; *i.e.*, transmitted packets from the sender to the receiver) at the beginning of a session, since the values returned from the DRL controller at that time are mostly related to the training dataset.

To overcome this possible inaccuracy, we compare the prediction results obtained from the DRL controller with those from the heuristic controller and validate their accuracy. To do so, we use symmetric mean absolute percentage error (sMAPE). First, we compute the absolute difference (Dif_t) between the predicted bandwidth values given by the heuristic controller ($Heuristicbw_t$) and the DRL controller ($DRLbw_t$). Second, we compute the average predicted bandwidth value (Avg_t) based on both controllers. If the output from the percentage ($\frac{Dif_t}{Avg_t}$) is equal to or more than 30%, the algorithm decides not to use the DRL controller and feeds the output of the heuristic controller to the DRL controller for later use. In time, the percentage between the outputs of the two controllers reduces and the DRL controller starts making a better prediction. The essential steps of the adaptive selector are highlighted in Algorithm 2. We note that this algorithm also monitors the difference between DRL and heuristic controllers in case of deviations (corner cases) from the expected converged predictions from both controllers. If a deviation happens, it switches back to the heuristic controller. However, we observed this situation only occasionally under some network conditions. Once the DRL controller starts performing well, it keeps doing so in the long run.

TABLE I: Average results in terms of sMAPE and total score for different percentage values of the adaptive selector.

%	5	10	20	25	**30**	40	50
sMAPE	0.75	0.68	0.27	0.18	**0.13** ↓	0.43	0.54
Total score	55	59	70	78	**93** ↑	61	60

We decided the threshold of 30% to switch between BoB controllers empirically. We performed extensive experiments to find a suitable percentage that resulted in high bandwidth prediction accuracy and good scores (defined in Section IV-C) in the long term (*i.e.*, the whole live video session). In particular, we ran many tests with various percentage values, from 5% up to 50%, using different network conditions (see Section III-A1) and video content (same as given Section IV). Table I, summarizes the outcome over all the tests. As one can see, the percentage of 30% achieves the best performance in terms of the lowest sMAPE and highest total score compared to the other percentages.

Algorithm 2 Adaptive Selector

```
 1: function ADAPTIVESELECTOR
 2:     for Each time window $W_t$, $W_t > 0$ do
 3:         $Heuristicbw_t \leftarrow$ HeuristicController($c_t, d_t, l_t, X_t^r$)
 4:         $DRLbw_t \leftarrow$ DRLController($c_t, d_t, l_t, X_t^r$)
 5:         $Dif_t = |DRLbw_t - Heuristicbw_t|$
 6:         $Avg_t = \frac{DRLbw_t + Heuristicbw_t}{2}$
 7:
 8:         if $\frac{Dif_t}{Avg_t} \geq 0.3$ then
 9:             $x_t^r \leftarrow Heuristicbw_t$
10:         else
11:             $x_t^r \leftarrow DRLbw_t$
12:         end if
13:     end for
14:     Return($x_t^r$)
15: end function
```

D. (Sender-Side) Loss-Based Controller

The sender and receiver controllers complement each other to select a suitable bitrate. The loss-based controller is located at the sender and is responsible for selecting the sending rate based on the packet loss ratio. At every time window W_t, the sender receives an RTCP feedback message from the receiver carrying the predicted bandwidth x_t^r and loss ratio l_t computed at the receiver. Based on this, the sender selects the sending rate x_t^s as follows:

$$x_t^s = \begin{cases} x_{t-1}^s \times (1 - 0.5 \times l_t), & l_t > 0.1; \\ 1.05 \times x_{t-1}^s, & l_t < 0.02; \\ x_{t-1}^s, & \text{otherwise.} \end{cases}$$

Here, the selected sending rate x_t^s changes depending on the loss ratio l_t where: (i) x_t^s remains constant in case l_t is small ($0.02 \leq l_t \leq 0.1$), (ii) x_t^s decreases multiplicatively in case l_t is high ($l_t > 0.1$), and (iii) x_t^s increases multiplicatively in case l_t is very small ($l_t < 0.02$). The final selected sending rate is then computed as follows: $x_t^\star = \min(x_t^r, x_t^s)$. This value $x_t^\star$ is provided to the encoder as the target bitrate. The chosen loss ratio ranges are given by GCC, as referenced from [22].

E. Parameter Choices and Training Setup

We fixed α and β at 0.85 and 1.08, respectively, in the BoB delay-based controller. These values have been empirically tuned based on our experiments and our finding is also aligned with [22]. For the BoB DRL controller, training parameters can impact its performance, so we empirically set the parameters as follows: the maximum number of episodes N to 2,000, the policy update interval T_θ to 4,000 time windows, the PPO k-steps to 20, the PPO clip parameter to 0.2, the discount factor γ to 0.99, the Adam learning rate lr to 3×10^{-5}, the Adam β to 0.999, the number of recent samples n to eight, and the time window W_t during which the states are captured to 200 ms. To train our DRL model, we used around 500 network traces in total from different datasets: The ACM MMSys'21 grand challenge on bandwidth estimation in RTC dataset [3], Belgium 4G/LTE [56], Norway 3G/HSDPA [50], NYU LTE [46], FCC [28], and Synthetic [12]. We randomized and divided them into two sets: 80% for BoB training and 20% for BoB testing. With 80–20 train-test split, we performed 5-fold walk-forward cross-validation on each dataset. The training output is one DRL model with `.pth` extension, which we use for the online inference and our results are presented in Section IV.

F. BoB Implementation and Challenges

To implement BoB, we used the platform, named AlphaRTC [5], provided by Microsoft's grand challenge on RTC [3] that comprises two main parts: offline training and online testing.

1) Offline Training: The trace-driven simulator mainly uses PyTorch v1.10 [47] for the deep reinforcement learning components and implements the GYM for a typical RTC system. The GYM uses ns-3 and WebRTC applications to simulate a sender-receiver RTC environment. The BoB model training uses real-world network traces to simulate the network conditions between the sender and receiver in terms of available bandwidth, RTT and packet loss.

2) Online Testing: The AlphaRTC [5] framework is a fork of Google's WebRTC project with machine learning-based bandwidth estimation. We use this framework and plug in the BoB bandwidth predictor for RTC system testing using real-world traces. The BoB controller is implemented in Python and consists of about 1,700 lines of new code, available online at [11]. In this code, the BoB controller is implemented as a class under file name BandwidthEstimator_bob.py, which comprises of three functions: AdaptiveSelector(), HeuristicController() and DRLController().

3) Challenges: It is well known [66], [69] that any DRL model requires significant data to converge to the best bandwidth accuracy prediction because of the training-to-testing gap issue [66]. Achieving the best bandwidth prediction requires a ramp-up during the live video session, which might hinder the overall performance of an RTC system. For example, during the design of BoB, we tried to use the DRL controller from the beginning of the video session, but we observed that our model experienced frequent bandwidth underprediction issues, which adversely impacted its convergence during the session. We also observed that the underprediction remained for some time because the penalties (loss, delay) were close to zero due to the initial scarcity of data history, so the model thought it was performing well. This is also one of the known issues with a DRL model, which trains an agent by giving it feedback (QoE; a combination of the receiving rate, delay and loss) for decisions while interacting with an environment. Therefore, this confirms why a DRL model requires some time to converge to the best bandwidth decisions. To avoid these issues, we used the heuristic controller to perform the bandwidth prediction decisions and at the same time collected enough data, allowing fast convergence of the DRL controller to the best decisions once the selector switched to it.

For all considered network traces, we found that the BoB DRL model requires on average 500 training episodes to converge to the best bandwidth prediction decisions. In the testing phase, we found that at the beginning of a video session, the BoB DRL model requires at most 10–15 seconds of packet transmissions based on the heuristic controller to start performing well (refer to Section III-C3). This is important at the beginning of a live video session as the DRL model requires the latest eight bandwidth prediction values as one of the input channels of the NN. These values are given by the heuristic controller. Since the heuristic controller is purely based on the last value of the heuristic such as packet loss or delay to make bandwidth prediction decisions, it requires minimal data. However, the heuristic controller cannot easily be generalized to various network conditions as it heavily depends on some hardcoded configuration parameters. As a result, it might suffer from inaccurate bandwidth predictions under some network conditions. This motivates and confirms the hybrid selection choice for BoB.

IV. Performance Evaluation

In this section, we evaluate the effectiveness of BoB against the purely heuristic-based (GCC) approach and the latest hybrid (heuristic and learning-based) approaches proposed for RTC systems including Gemini [55] and HRCC [57]. Our evaluation is divided into two setups: emulation-based and Internet-based.

A. Evaluation Setups

1) Emulation-Based Setup: In order to evaluate the effectiveness of BoB over an end-to-end controlled system, we used a physical machine running Ubuntu 18.04 LTS OS with dual 20-core Intel E5-2630 v4 @ 2.20GHz processors and 192 GB memory. We ran the trace-driven framework (AlphaRTC) in an isolated environment using the Docker container provided by the Microsoft team and we installed extra library dependencies for the tc [4] command to be able to throttle the bandwidth between the sender and receiver and introduce packet loss/delay following the network profiles highlighted in Fig. 5. The Estimator class, contained in the source file Bandwidth Estimator.py, is used by the Docker environment to call the desired bandwidth estimator (BoB, HRCC or Gemini) and the get_estimated_bandwidth() method of the Estimator class is invoked as packets arrive in the setup. Each solution logs the predicted bandwidth values, bandwidth prediction accuracy and error, receiving rate score, delay score, packet loss score, network score, video score and total score.

▷ **Network Profiles.** The network profiles we used in the evaluation, re-purposed for this work from [15], are shown in Fig. 5. The profiles are extracted randomly from 20% of network traces assigned for testing, namely: LTE, Twitch, Cascade, FCC Amazon and Synthetic. For FCC Amazon and Synthetic, we fixed the delay to 50 ms and loss to 0.08%.

▷ **Video Sample.** For the video sample, we used the *Big Buck Bunny* video sample [2] with 24 fps and 640×360 pixel resolution that was approximately one minute long. The simulation configuration files are given in [5], including receiver_pyinfer.py and sender_pyinfer.py and updated with the test video source and properties. In these files, there is also an *autoclose* parameter that specifies the duration (in seconds) of the system test to be performed. In our simulations, this parameter was set to 60 seconds.

2) *Internet-Based Setup:* OpenNetLab provides an Internet-based public testbed (https://opennetlab.org/) that creates a unified measuring platform to validate the performance of RTC-based solutions, including BoB, under unseen network conditions in the wild through initiating several end-to-end RTC calls. This testbed includes wired, wireless and mobile networks and heterogeneous nodes with support from universities throughout Asia. The nodes are in China (Beijing, Hefei, Nanjing, Lanzhou, Shenzhen and Hong Kong), South Korea (Seoul and Daejeon) and Singapore (Queenstown). The set of nodes in the testbed is coordinated using Azure Backend microservices.

To test the end-to-end RTC calls with the BoB solution versus competitors (heuristic-based, Gemini and HRCC) over the Internet, we submitted a performance validation job by uploading the BoB trained DRL model and algorithms. We also specified the predefined resource (compute node and network type) and predefined scenarios (A, B and C) via a Web-based frontend. For each video sample, each scenario was run five times in a round-robin manner. These scenarios are highlighted in Table II and Fig. 6 shows the setup of the testbed. To validate the performance of BoB's competitors, we used the same process.

▷ **Network Profiles.** The public Internet-based testbed offers three types of network characteristics: High, Medium and Low Bandwidth (BW). The details of each network are highlighted in Table II.

▷ **Video Sample.** Each scenario (Table II) runs with

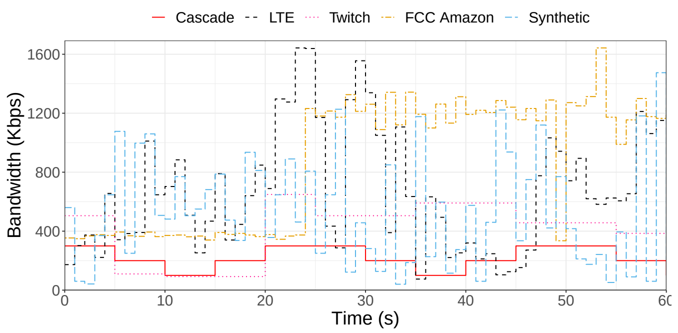


Fig. 5: The network profiles used in the simulations.

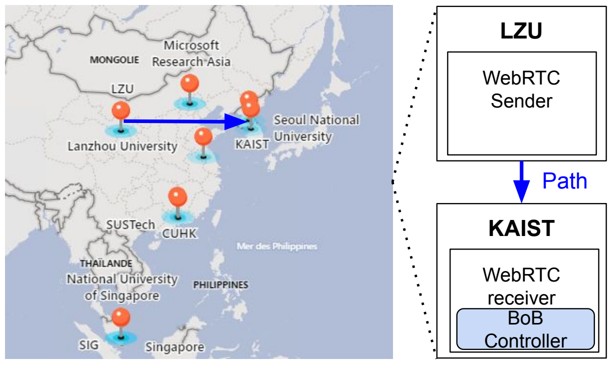


Fig. 6: An example of the OpenNetLab public Internet testbed.

different types of video samples including animation, movie, conversation, presentation and screen sharing over a remote desktop. Each video is five minutes long with various sets of frame rates (fps) and resolutions.

B. Comparisons

We compared BoB against three approaches: the heuristic approach, the winner (Gemini [55]) and runner-up (HRCC [57]) of the ACM MMSys'21 grand challenge on bandwidth estimation in RTC, organized by Microsoft. We selected Gemini and HRCC because (*i*) they represent the latest solutions using a hybrid approach, (*ii*) they are the winner and runner-up of the grand challenge, and (*iii*) their implementations are available in AlphaRTC, which allows us to replicate their claimed results.

C. Evaluation Metrics

We tested the efficiency of BoB and other approaches using the following evaluation metrics:

1) *Bandwidth Prediction Error and Accuracy:* The bandwidth prediction error and accuracy are calculated based on symmetric mean absolute percentage error (sMAPE). The sMAPE is an accuracy measure based on percentage (or relative) errors between predicted bandwidth values (x_t) and actual network profile values (y_t) for the total samples T, and its function is given as follows:

$$\text{sMAPE} = \frac{1}{T} \times \sum_{t=t}^{T} \frac{|y_t - x_t|}{(y_t + x_t)/2}, \text{accuracy} = (1 - \frac{\text{sMAPE}}{2}) \times 100$$

TABLE II: Scenarios for the public Internet tesbed.

	Network Profile	Sender Node	Receiver Node	Path Bandwidth	Avg. RTT
A	High Bandwidth	Lanzhou (Wired)	Seoul (Wired)	$>$ 100 Mbps	30 ms
B	Medium Bandwidth	Beijing (Mobile)	Hong Kong (Wired)	2-3 Mbps	62 ms
C	Low Bandwidth	Beijing (Weak wireless)	Hong Kong (Wired)	$<$ 1 Mbps	55 ms

2) Network Score: The network score (denoted by $\mathcal{N}_s$) is computed as a combination of three metrics: delay score (d_s), loss score (l_s) and receiving rate score (c_s), as follows:

$$\mathcal{N}_s = w_1 \times d_s + w_2 \times c_s + w_3 \times l_s,$$

where

$$\begin{aligned} d_s &= 100 \times \frac{max\_delay - delay\_95^{th}}{max\_delay - min\_delay}, \\ c_s &= 100 \times \frac{c}{ground\_truth\_c}, \text{and} \\ l_s &= 100 \times (1 - l). \end{aligned}$$

Here, $w_1 = w_2 = 0.1$ and $w_3 = 0.5$ are the weights of the network score. The *max_delay* is fixed to 400 ms and *min_delay* is the minimum delay achieved during the RTC session. The *ground_truth_c* refers to the corresponding average bandwidth that can be obtained in an ideal environment (such as when there is no loss and no delay). Since we have the network profiles for the experiments, it is easy to compute *ground_truth_c*, which is fixed as the overall average actual bandwidth value in each corresponding network profile (Cascade: 220 Kbps, LTE: 741 Kbps, Twitch: 335 Kbps, FCC Amazon: 676 Kbps, Synthetic: 581 Kbps). Finally, l is the packet loss ratio.

3) Video Score: The video score (denoted by $\mathcal{V}_s$) is calculated with respect to video perpetual quality based on Video Multi-Method Assessment Fusion (VMAF)[2] as follows:

$$\mathcal{V}_s = 100 \times vmaf\_score,$$

where *vmaf_score* is the average VMAF value (ranges between 0 and 1) computed based on per-frame VMAF values resulting from the source and encoded video.

4) Total Score: The total score (denoted by $\mathcal{T}_s$) is computed as a combination of $\mathcal{N}_s$ and $\mathcal{V}_s$, as follows:

$$\mathcal{T}_s = \mathcal{N}_s + w_4 \times \mathcal{V}_s,$$

where w_4 is the weight factor associated with the video score which is fixed to 0.3, and $\sum_{i=1}^{4} w_i = 1$. We note that the network, video and total score formulation was originally supplied by the Microsoft grand challenge organizers [6]. These scores cover all the main metrics to evaluate the QoE performance of an RTC system, which are widely used in many papers such as [55], [66], [69], [27]. For instance, the video score uses VMAF, the widely used metric proposed by Netflix to compute video perceptual quality, while the network score combines the important metrics for an RTC system including packet loss, delay and receiving rate.

[2]Available [Online]: https://github.com/Netflix/vmaf

D. Results and Analysis

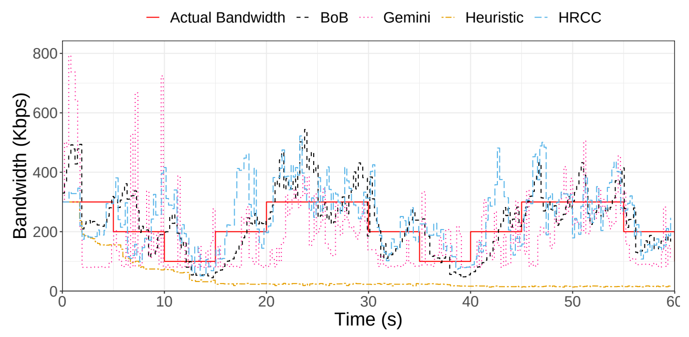
We now compare and describe the performance of different solutions. For statistically meaningful results, we repeated all experiments five times for each solution with the same configuration and all the presented results show the averages over the five runs. We divided our results into two setups: emulation-based and Internet-based.

1) Emulation-Based Results: First, we analyze the performance in terms of bandwidth prediction accuracy that each solution achieves. Then, we compare the performance of different solutions in terms of network, video and total scores, expressed with their metrics.

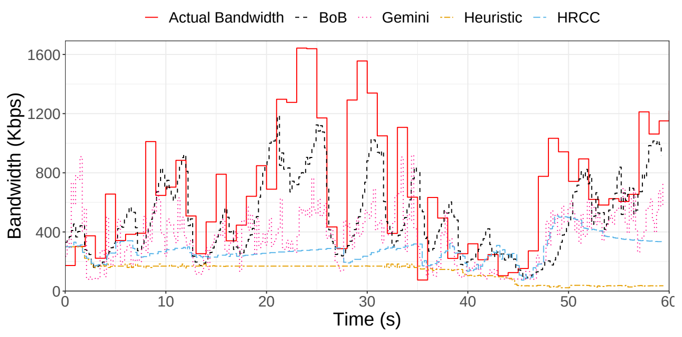
▷ **Bandwidth Prediction Accuracy.** The time series plots for different solutions for every network profile are depicted in Fig. 7. The overall average bandwidth prediction accuracy and prediction error in terms of sMAPE are provided in the first two columns of Table III. The red solid lines in Fig. 7 represent the actual bandwidth for the network profiles. A superior solution must determine a bandwidth within a close proximity of these solid lines. Overall, we notice that BoB achieves the best bandwidth prediction accuracy (and the lowest prediction error). Specifically, BoB improves the overall average bandwidth prediction accuracy by 67.63% (Cascade: 61.72%, LTE: 41.71%, Twitch: 81.64%, FCC Amazon: 73.27%, Synthetic: 79.80%), and reduces the overall average bandwidth prediction error by 49.11% (Cascade: 38.95%, LTE: 46.45%, Twitch: 62.14%, FCC Amazon: 71.07%, Synthetic: 26.94%) compared to the other solutions across all the network profiles. However, only in the Cascade profile, HRCC is slightly better than BoB in terms of the average bandwidth prediction accuracy with a marginal improvement of 0.19%. Therefore, HRCC was able to achieve a better receiving rate, network, video and total scores compared to BoB in the Cascade profile.

Looking at the results further, BoB is able to achieve a higher average receiving rate score of 73.64%, compared to Gemini (45.28%), HRCC (54.77%) and Heuristic (31.78%) across all the network profiles. This comes at a price of a smaller delay score and a comparable loss score for BoB in each network profile. In fact, BoB's formulation strives to ensure a good trade-off between these conflicting metrics (*i.e.*, receiving rate, delay and packet loss). Such a trade-off by BoB is confirmed through achieving higher network, video and total scores with an improvement of 35.85% (Cascade: 29.21%, LTE: 31.53%, Twitch: 58.00%, FCC Amazon: 45.77%, Synthetic: 14.74%), 5.90% (Cascade: 6.41%, LTE: 4.59%, Twitch: 6.73%, FCC Amazon: 7.40%, Synthetic: 4.37%), and 23.03% (Cascade: 18.38%, LTE: 18.95%, Twitch: 36.19%, FCC Amazon: 31.49%, Synthetic: 10.12%), respectively, compared to its competitors across all five network profiles.

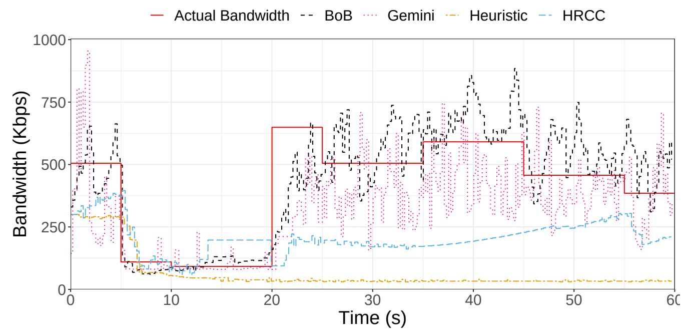
Compared to BoB, we also observe that other solutions generally suffer from either bandwidth overprediction or underprediction due to their designs. As shown in Fig. 7, Gemini tends to underutilize the bandwidth, which expectedly produces a low receiving rate score but also a higher packet



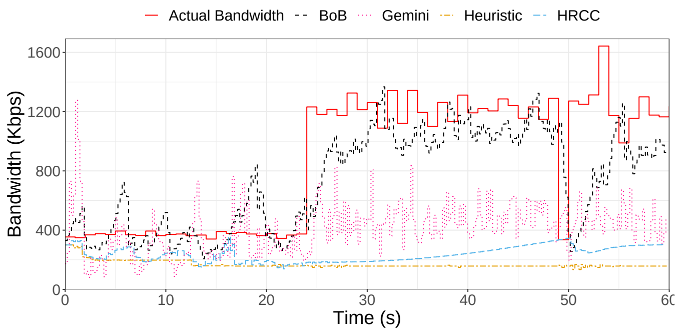
(a) Profile: Cascade



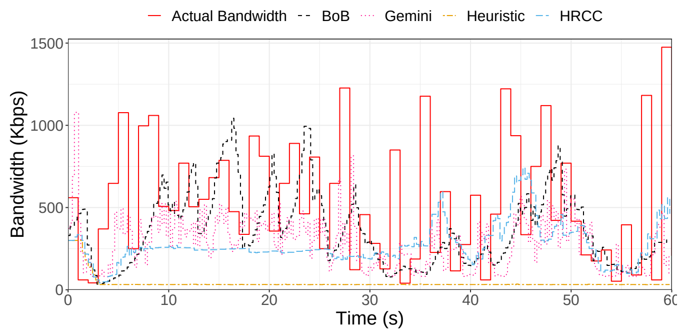
(b) Profile: LTE



(c) Profile: Twitch



(d) Profile: FCC Amazon



(e) Profile: Synthetic

Fig. 7: Actual and predicted bandwidth for different network profiles.

delay and loss score than BoB and HRCC. This happens because Gemini fails to timely switch between the learning and heuristic-based prediction. For example, for the FCC Amazon profile (see Fig. 7d), the learning-based prediction for Gemini fails to track the increase in the actual bandwidth. Similarly, HRCC generally fails to learn suitable parameters for the heuristic-based algorithm, which leads to a bandwidth underprediction issue for various network profiles, which is most visible in Figs 7b, 7c and 7d. As a result, HRCC suffers from poor video quality. This also confirms that HRCC is more suitable for more stable, low-bandwidth scenarios.

One interesting observation is that the Heuristic solution is not able to recover from bandwidth underestimations during the whole RTC session, which contributes to poor video quality (see the score results in Table III). This outcome confirms the difficulty and importance of bandwidth prediction in RTC [3], and also shows how urgent it is to have a hybrid solution that combines learning- and heuristic-based algorithms. In contrast, BoB, harmoniously fuses both

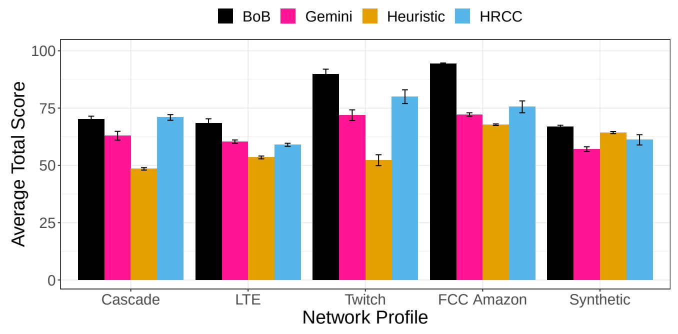


Fig. 8: Average total score for different network profiles (emulation-based).

algorithms and tries to predict the bandwidth within a small margin of its actual value during the RTC session, and works equally well across different network profiles.

▷ **Scores and Their Metrics.** We evaluate different solutions in terms of network, video and total scores and their metrics

TABLE III: Average simulation results for different network profiles (↑: higher is better, ↓: lower is better).

	Avg. Prediction Accuracy (%)	Avg. Prediction Error (sMAPE)	Avg. Receiving Rate Score (%)	Avg. Delay Score (%)	Avg. Loss Score (%)	Avg. Network Score (%)	Avg. Video Score (%)	Avg. Total Score (%)
Cascade								
BoB	84.89	**0.30** ↓	63.26	15.71	95.45	42.75	91.38	70.16
Gemini	74.85	0.50	46.56	35.78	97.64	36.62	87.73	62.94
HRCC	**85.06** ↑	**0.30** ↓	**63.91** ↑	21.27	91.69	**43.25** ↑	**92.23** ↑	**70.92** ↑
Heuristic	31.22	1.38	22.36	**37.22** ↑	**99.38** ↑	24.84	78.79	48.48
LTE								
BoB	**78.94** ↑	**0.42** ↓	**56.98** ↑	32.05	88.57	**40.55** ↑	**92.59** ↑	**68.33** ↑
Gemini	68.07	0.64	33.35	71.31	99.01	33.71	88.85	60.36
HRCC	63.65	0.73	35.77	38.00	97.51	31.44	91.82	58.98
Heuristic	42.63	1.15	20.39	**77.56** ↑	**99.66** ↑	27.92	85.17	53.47
Twitch								
BoB	**88.84** ↑	**0.22** ↓	**94.87** ↑	52.11	92.29	**61.88** ↑	**92.80** ↑	**89.72** ↑
Gemini	82.85	0.34	60.39	45.24	**98.49** ↑	44.57	91.12	71.91
HRCC	65.92	0.68	74.74	**54.57** ↑	97.06	52.53	91.50	79.98
Heuristic	29.33	1.41	31.77	29.81	95.98	28.46	79.36	52.27
FCC Amazon								
BoB	**86.60** ↑	**0.27** ↓	**100** ↑	60.02	97.17	**65.72** ↑	**95.19** ↑	**94.28** ↑
Gemini	63.91	0.72	55.09	82.56	99.71	45.77	88.14	72.21
HRCC	50.42	0.99	60.36	79.52	99.84	48.11	91.46	75.55
Heuristic	40.74	1.19	44.91	**93.72** ↑	**100** ↑	41.83	86.45	67.76
Synthetic								
BoB	**65.88** ↑	**0.68** ↓	**53.08** ↑	28.58	96.82	**39.08** ↑	**92.59** ↑	**66.85** ↑
Gemini	62.15	0.76	30.99	65.77	97.94	31.87	84.08	57.09
HRCC	60.43	0.79	39.06	42.28	97.56	33.52	92.11	61.15
Heuristic	20.31	1.59	39.49	**75.32** ↑	**99.42** ↑	37.22	90.35	64.32

(see Section IV-C). The average total scores are given in Fig. 8 and the individual metrics are tabulated in Table III. Fig. 8 shows that BoB has the highest performance in the LTE, Twitch, FCC Amazon and Synthetic network profiles.

For the Cascade profile, BoB and HRCC perform similarly in terms of the total score and average prediction error but differ in terms of delay and loss scores. BoB can cause increased delays without significantly increasing the packet loss, whereas HRCC has less delay but more packet loss. At the end of the RTC session, the network scores were quite close with these different trade-offs. In the LTE profile, BoB uses higher receiving rates with a delay and loss cost, which still results in a better video score.

In the Twitch profile, BoB has the highest average bandwidth prediction accuracy, again resulting in high bitrates without inducing much delay and loss. The reason is that BoB can upshift fast and utilize the available bandwidth after the first 20 seconds (which confirms the convergence of the learning-based algorithm to the optimal solution), where both Gemini and HRCC still underpredict the bandwidth most of the time.

Fig. 7d illustrates that BoB has the best fit for the actual bandwidth values, especially after the 25^{th} second. As a result of its accurate bandwidth prediction, BoB achieves the highest receiving rate score and a loss score slightly lower than the best one achieved by the other solutions. Synthetic is one of the most challenging profiles as it exhibits fast and sudden changes in the bandwidth. Even if the bandwidth is changing frequently, the prediction error is the smallest with BoB. Moreover, BoB achieves the highest receiving rate score.

Overall, BoB achieves the smallest average prediction error with a value of 0.38 and the highest average prediction accuracy with a value of 81.03%. As for the Heuristic, the prediction accuracy is the worst yet it has the highest delay and loss scores. The overall indicators imply that there is further room for improvement in RTC systems, where the bandwidth prediction and bitrate selection should be jointly considered to achieve better application performance, *i.e.*, to use the full available bandwidth for the media without inducing significant packet delay or loss under diverse network conditions.

▷ **Results Summary.** In all the considered experiments, BoB performs better in most performance metrics and outperforms Gemini, HRCC and Heuristic solutions under various network conditions. This is mainly due to BoB's design that combines heuristic and learning-based controllers for bandwidth prediction and bitrate selection for RTC systems. The percentages of improvement (%) achieved by BoB versus other solutions are calculated by comparing BoB's results with the ones obtained by each solution. The results are summarized in Table IV.

TABLE IV: Summary of the average results. Percentage improvements of BoB over the other solutions, at scale.

BoB vs.	Avg. Prediction Accuracy (%)	Avg. Network Score (%)	Avg. Video Score (%)	Avg. Total Score (%)
Gemini	15.62	28.42	5.67	19.42
HRCC	27.87	19.76	1.19	12.21
Heuristic	159.39	59.37	10.84	37.45
Average	67.63	35.85	5.90	23.03

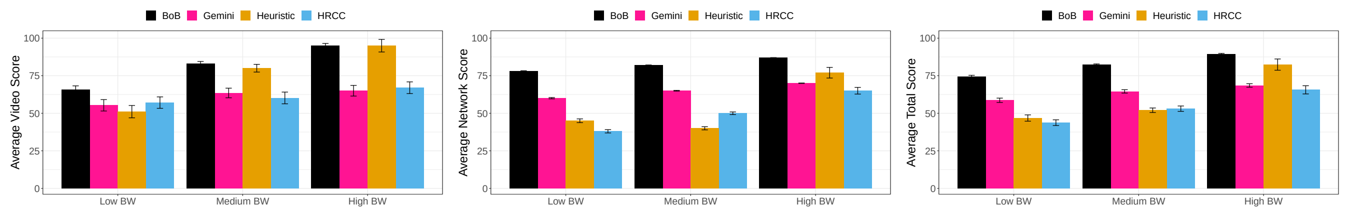


Fig. 9: Average video, network and total scores for different scenarios (Interned-based).

E. *Internet-Based Results*

We further validate the performance of BoB against its competitors in terms of network, video and total scores through the OpenNetLab public Internet-based testbed. Fig. 9 shows the average scores for different scenarios. First, BoB achieves the highest scores (video, network and total) compared to Heuristic, Gemini and HRCC in all scenarios. This demonstrates and validates the capabilities of BoB in adapting to unseen network conditions. Second, HRCC suffers from a low delay score, while Heuristic suffers from a low loss score in a low bandwidth network. On the contrary, HRCC suffers from a low loss score, while Heuristic suffers from a low delay score in a medium bandwidth network. Such an outcome confirms their low network scores in low and medium bandwidth scenarios. Third, Gemini is the runner-up to BoB, however, it does not perform well in the high bandwidth scenario. Overall, the results here are quite similar to the ones obtained in the controlled emulation-based experiments. Specifically, BoB provides the following improvements over Gemini, HRCC and Heuristic, respectively:

- Scenario A: Network score by (29.89%, 105.26%, 73.33%), video score by (18.80%, 15.06%, 28.65%) and total score by (26.75%, 69.93%, 58.71%).
- Scenario B: Network score by (26.15%, 64%, 105%), video score by (30.71%, 37.87%, 3.75%) and total score by (27.50%, 55.11%, 58.27%).
- Scenario C: Network score by (24.28%, 33.85%, 12.99%), video score by (46.46%, 42.09%, 0.21%) and total score by (30.60%, 36.37%, 8.59%).

V. Discussion and Open Directions

To inspire further work in this area, we discuss three interesting future research directions in RTC systems.

1) We believe that QoE metrics and bandwidth prediction accuracy should be jointly optimized for better performance in RTC systems. BoB aims to achieve this objective but leaves room for improvement under more complex network conditions and RTC-based application requirements.
2) Various bandwidth prediction models may perform differently based on which metrics they tend to prioritize or sacrifice, which makes comparisons between these models and drawing conclusions difficult, especially if their scores are similar. One way to compare them is to find the best and worst performing model in each QoE metric and quantify the relative performance of the other models against these boundaries or targets so that a system implementer may choose a scheme based on his/her own priorities and preferences. Note that the QoE is a compound metric and if the aggregated values are similar, then the individual components (latency, bandwidth variations, *etc.*) can be examined to compare different prediction models.
3) Fairness is an important aspect when deploying a solution on the Internet where usually competition exists between different streams for the available bandwidth in a shared network environment (either on the server or the client side). This competition can be between intra (*e.g.*, between different RTC streams) or inter traffic (*e.g.*, between RTC and non-RTC streams like HTTP-based streaming traffic). We believe that analyzing fairness and building a fairness-aware solution is critical for optimizing the QoE. Also, the designed solution should consider the impact and diversity in transport-layer congestion control protocols (BBR, NewReno, Cubic, NADA, SCReAM, *etc.*). Note that the definition of fairness deserves some examination, too. For example, is it fair to treat a small phone (small screen and likely one viewer) and a large big-screen TV (large screen and likely more than one viewer) the same [10]?

VI. Conclusions

We developed a receiver-side hybrid bandwidth predictor for RTC services in this study, named BoB. Hybrid prediction is achieved using a heuristic and a learning-based controller. The heuristic uses a delay filter, while the learning-based mechanism uses DRL actor-critic networks with PPO and an Adam optimizer for model training and policy updates. To perform the bandwidth prediction task, BoB uses the heuristic-based controller at the beginning of each session and then switches to the learning-based controller for more accurate bandwidth prediction. As a result, BoB can achieve a higher receiving rate with reduced packet delay and loss ratio, contributing to a better user experience. During each fixed time window, BoB collects packet-level data, including the receiving rate, packet delay, packet loss and the last eight predicted bandwidth values as the input state into the neural network to predict the bandwidth for the next time epoch.

BoB has been integrated into AlphaRTC and the results show the superiority of BoB for bandwidth prediction in RTC. BoB achieves up to 15.62% and 27.87% better bandwidth prediction accuracy than Gemini and HRCC (the winning and runner-up solutions, respectively, in the ACM MMSys'21 grand challenge), respectively, under various challenging network conditions. For future work, we plan to implement FEC techniques using DRL and perform larger scale real-world experiments.

Acknowledgements

This work was supported in part by Singapore Ministry of Education Academic Research Fund Tier 2 under MOE's official grant number T2EP20221-0023 and by the Scientific and Technological Research Council of Turkey under grant number 120C154.

REFERENCES

[1] A Google Congestion Control Algorithm for Real-Time Communication. [Online] Available: https://datatracker.ietf.org/doc/html/draft-ietf-rmcat-gcc-02. Accessed on Jan. 21, 2022.
[2] Big Buck Bunny Video. [Online] Available: https://download.blender.org/peach/bigbuckbunny_movies/BigBuckBunny_320x180.mp4. Accessed on Jan. 21, 2022.
[3] Grand Challenge on Bandwidth Estimation for Real-Time Communications. [Online] Available: https://2021.acmmmsys.org/rtc_challenge.php. Accessed on Jan. 21, 2022.
[4] iproute2. [Online] Available: https://wiki.linuxfoundation.org/networking/iproute2. Accessed on Jan. 21, 2022.
[5] OpenNetLab AlphaRTC. [Online] Available: https://github.com/OpenNetLab/AlphaRTC. Accessed on Jan. 21, 2022.
[6] RTC Evaluation Score. [Online] Available: https://github.com/OpenNetLab/challenge-HOWTO. Accessed on Jan. 21, 2022.
[7] S. Abbasloo, C.-Y. Yen, and H. J. Chao. Classic meets modern: A pragmatic learning-based congestion control for the internet. In *ACM SIGCOMM*, 2020.
[8] V. Arun and H. Balakrishnan. Copa: Practical delay-based congestion control for the internet. In *USENIX NSDI*, 2018.
[9] T. Balan, A. Stanciu, F. Sandu, and S. Surariu. Webrtc based elearning platform. *eLearning & Software for Education*, 2017.
[10] A. C. Begen. Spending quality time with the web video. *IEEE Internet Comput.*, 20(6):42–48, Nov./Dec. 2016 (DOI: 10.1109/MIC.2016.49).
[11] A. Bentaleb, M. N. Akcay, M. Lim, A. C. Begen, and R. Zimmermann. BoB Code. [Online] Available: https://github.com/NUStreaming/BoB. Accessed on Aug. 21, 2022.
[12] A. Bentaleb, A. C. Begen, S. Harous, and R. Zimmermann. Want to play DASH? a game theoretic approach for adaptive streaming over HTTP. In *ACM MMSys*, 2018 (DOI: 10.1145/3204949.3204961).
[13] A. Bentaleb, A. C. Begen, S. Harous, and R. Zimmermann. Data-driven bandwidth prediction models and automated model selection for low latency. *IEEE Trans. Multimedia*, 23:2588–2601, 2021 (DOI: 10.1109/TMM.2020.3013387).
[14] A. Bentaleb, A. C. Begen, and R. Zimmermann. SDNDASH: Improving QoE of HTTP adaptive streaming using software defined networking. In *ACM Multimedia*, 2016 (DOI: 10.1145/2964284.2964332).
[15] A. Bentaleb, M. N. Akcay, M. Lim, A. C. Begen, and R. Zimmermann. Catching the moment with lol$^+$ in twitch-like low-latency live streaming platforms. *IEEE Trans. Multimedia*, 24:2300–2314, 2022.
[16] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann. A survey on bitrate adaptation schemes for streaming media over HTTP. *IEEE Communications Surveys & Tutorials*, 21(1):562–585, Firstquarter 2019 (DOI: 10.1109/COMST.2018.2862938).
[17] N. Bouten, S. Latré, J. Famaey, W. Van Leekwijck, and F. De Turck. In-network quality optimization for adaptive video streaming services. *IEEE Trans. Multimedia*, 16(8):2281–2293, 2014.
[18] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP vegas: New techniques for congestion detection and avoidance. In *ACM SIGCOMM*, SIGCOMM '94, page 24–35, New York, NY, USA, 1994.
[19] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. Bbr: congestion-based congestion control. *Communications of the ACM*, 60(2):58–66, 2017.
[20] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis, and V. Jacobson. BBR v2 a Model-based Congestion Control. [Online] Available: https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00, 2019. Accessed on Jan. 21, 2022.
[21] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo. Analysis and design of the google congestion control for web real-time communication (webrtc). In *ACM MMSys*, 2016.
[22] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo. Congestion control for web real-time communication. *IEEE/ACM Trans. Networking*, 25(5):2629–2642, 2017.
[23] Cisco. VNI Complete Forecast Highlights. [Online] Available: https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_2021_Forecast_Highlights.pdf, Oct. 2020. Accessed on Jan. 21, 2022.
[24] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira. {PCC}: Re-architecting congestion control for consistent high performance. In *USENIX NSDI*, 2015.
[25] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira. PCC vivace: Online-learning congestion control. In *USENIX NSDI*, 2018.
[26] Y. Dong, L. Song, R. Xie, and W. Zhang. An elastic system architecture for edge based low latency interactive video applications. *IEEE Trans. Broadcasting*, 2021.
[27] J. Fang, M. Ellis, B. Li, S. Liu, Y. Hosseinkashi, M. Revow, A. Sadovnikov, Z. Liu, P. Cheng, S. Ashok, et al. Reinforcement learning for bandwidth estimation and congestion control in real-time communications. *arXiv preprint arXiv:1912.02222*, 2019.
[28] FCC. Raw Data - Measuring Broadband America. [Online] Available: https://goo.gl/gJLND4, 2016. Accessed on Jan. 21, 2022.
[29] S. Floyd, T. Henderson, and A. Gurtov. The newreno modification to TCP's fast recovery algorithm. [Online] Available: https://datatracker.ietf.org/doc/html/rfc3782, 2004. Accessed on Jan. 21, 2022.
[30] S. Fouladi, J. Emmons, E. Orbay, C. Wu, R. S. Wahby, and K. Winstein. Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol. In *USENIX NSDI*, pages 267–282, 2018.
[31] S. Ha, I. Rhee, and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.
[32] R. Herrero. Integrating hec with circuit breakers and multipath rtp to improve rtc media quality. *Telecommunication Systems*, 64(1):211–221, 2017.
[33] R. Hong, Q. Shen, L. Zhang, and J. Wang. Continuous bitrate & latency control with deep reinforcement learning for live video streaming. In *ACM Multimedia*, 2019.
[34] T. Huang, R. Zhang, and L. Sun. Zwei: A self-play reinforcement learning framework for video transmission services. *IEEE Trans. Multimedia*, 2021.
[35] T. Huang, R.-X. Zhang, C. Zhou, and L. Sun. Qarc: Video quality aware rate control for real-time video streaming based on deep reinforcement learning. In *ACM Multimedia*, 2018.
[36] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *ACM SIGCOMM*, 2014.
[37] V. Jacobson. Congestion avoidance and control. *ACM SIGCOMM CCR*, 18(4):314–329, 1988.
[38] B. Jansen, T. Goodwin, V. Gupta, F. Kuipers, and G. Zussman. Performance evaluation of webrtc-based video conferencing. *ACM SIGMETRICS Performance Evaluation Review*, 45(3):56–68, 2018.
[39] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar. A deep reinforcement learning perspective on internet congestion control. In *Int. Conf. Machine Learning*. PMLR, 2019.
[40] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *ACM CoNEXT*, 2012.
[41] I. Johansson and Z. Sarker. Self-clocked rate adaptation for multimedia. [Online] Available: https://datatracker.ietf.org/doc/html/rfc8298, 2020. Accessed on Jan. 21, 2022.
[42] W. Li, H. Zhang, S. Gao, C. Xue, X. Wang, and S. Lu. Smartcc: A reinforcement learning approach for multipath tcp congestion control in heterogeneous networks. *IEEE Jour. Selected Areas in Communications*, 37(11):2621–2633, 2019.
[43] Y. Li, X. Wang, H. Liu, L. Pu, S. Tang, G. Wang, and X. Liu. Reinforcement learning based resource partitioning for improving responsiveness in cloud gaming. *IEEE Trans. Computers*, 2021.
[44] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. Resource management with deep reinforcement learning. In *ACM Wksp. Hot Topics in Networks*, 2016.
[45] H. Mao, R. Netravali, and M. Alizadeh. Neural adaptive video streaming with pensieve. In *ACM SIGCOMM*, 2017.
[46] L. Mei, R. Hu, H. Cao, Y. Liu, Z. Han, F. Li, and J. Li. Realtime Mobile Bandwidth Prediction Using LSTM Neural Network. In *Int. Conf. Passive and Active Network Measurement*. Springer, 2019.
[47] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.
[48] M. Polese, F. Chiariotti, E. Bonetto, F. Rigotto, A. Zanella, and M. Zorzi. A survey on recent advances in transport layer protocols. *IEEE Communications Surveys & Tutorials*, 21(4):3584–3608, 2019.

[49] Review 42. Incredible Facebook Messenger Statistics in 2021. [Online] Available: https://review42.com/resources/facebook-messenger-statistics, July 2021. Accessed on Jan. 21, 2022.

[50] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen. Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications. In *ACM MMSys*, 2013.

[51] G. F. Riley and T. R. Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

[52] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[53] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui. Cloud gaming: architecture and performance. *IEEE Network*, 27(4):16–21, 2013.

[54] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Trans. Networking*, 28(4):1698–1711, 2020.

[55] Y. Tianrun, W. Hongyu, H. Runyu, Y. Shushu, L. Dingwei, and Z. Jiaqi. Gemini: An ensemble framework for bandwidth estimation in web real-time communications. In *ACM MMSys*, 2021.

[56] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck. HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks. *IEEE Communications Letters*, 20(11):2177–2180, Nov. 2016.

[57] B. Wang, Y. Zhang, S. Qian, Z. Pan, and Y. Xie. A hybrid receiver-side congestion control scheme for web real-time communication. In *ACM MMSys*, 2021.

[58] K. Winstein and H. Balakrishnan. TCP ex machina: Computer-generated congestion control. *ACM SIGCOMM CCR*, 43(4):123–134, 2013.

[59] X. Xie and X. Zhang. POI360: Panoramic mobile video telephony over LTE cellular networks. In *ACM CoNEXT*, 2017.

[60] Q. Xu, S. Mehrotra, Z. Mao, and J. Li. Proteus: network performance forecast for real-time, interactive mobile applications. In *ACM MobiSys*, 2013.

[61] P. K. Yadav, A. Shafiei, and W. T. Ooi. Quetra: A queuing theory approach to dash rate adaptation. In *ACM Multimedia*, 2017.

[62] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein. Pantheon: the training ground for internet congestion-control research. In *USENIX ATC*, 2018.

[63] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *ACM SIGCOMM*, 2015.

[64] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg. Adaptive congestion control for unpredictable cellular networks. In *ACM SIGCOMM*, 2015.

[65] P. Zaveri and S. Gould. Remote Work Boom. [Online] Available: https://bit.ly/36dWM82, July 2021. Accessed on Jan. 21, 2022.

[66] H. Zhang, A. Zhou, J. Lu, R. Ma, Y. Hu, C. Li, X. Zhang, H. Ma, and X. Chen. Onrl: improving mobile video telephony via online reinforcement learning. In *ACM MobiCom*, 2020.

[67] S. Zhang, W. Lei, W. Zhang, Y. Zhan, and H. Li. An online learning based path selection for multipath real-time video transmission in overlay network. *Trans. Emerging Telecommunications Technologies*, 31(11):e4131, 2020.

[68] Y. Zhang, S. Kwong, and S. Wang. Machine learning based video coding optimizations: A survey. *Information Sciences*, 506:395–423, 2020.

[69] A. Zhou, H. Zhang, G. Su, L. Wu, R. Ma, Z. Meng, X. Zhang, X. Xie, H. Ma, and X. Chen. Learning to coordinate video codec with transport protocol for mobile video telephony. In *ACM MoBiCom*, 2019.

[70] X. Zhu, P. Pan, M. Ramalho, and S. Mena. Network-assisted dynamic adaptation (NADA): A unified congestion control scheme for real-time media. [Online] Available: https://datatracker.ietf.org/doc/html/rfc8698, 2020. Accessed on Jan. 21, 2022.

Abdelhak Bentaleb received his Ph.D. in computer science from National University of Singapore (NUS), Singapore, in 2019. He continued as a research fellow at the same department until 2022. He is currently an assistant professor with the Department of Computer Science and Software Engineering, Concordia University, Canada. He is a Co-Founder of Atlastream Inc., Singapore. He got many prestigious awards like SIGMM Award for Outstanding PhD Thesis Award, DASH-IF Best PhD Dissertation Award and Dean's Graduate Research Excellence Award AY2018/2019. His research interests include applied AI in multimedia systems and communication, video streaming architectures, content delivery, distributed computing, computer networks and protocols, wireless communications, and mobile networks. Further information can be found at https://www.concordia.ca/ginacody/computer-science-software-eng/faculty.html?fpid=abdelhak-bentaleb.

Mehmet N. Akcay received his Ph.D. in computer science from Ozyegin University, at 2022. He received his B.Sc., in the field of Computer Engineering, from Istanbul Technical University in 2005. He completed his M.Sc. in the same field in Bogazici University in 2008 and he has been working in the industry for more than 10 years. His research interests are HTTP adaptive streaming, low-latency live streaming and software verification using formal methods.

May Lim is currently doing her Ph.D. in computer science at National University of Singapore (NUS), Singapore. She received her B.E.Sc. and M.Sc. from Nanyang Technological University (NTU), Singapore, in 2015. Her current research interest is primarily in multimedia streaming systems and she has done several works relating to low-latency streaming for live 2D and 6DoF videos.

Ali C. Begen (S'98–M'07–SM'12) has been a research and development engineer since 2001, and has broad experience in mathematical modeling, performance analysis, optimization, standards development, intellectual property and innovation. Between 2007 and 2015, he was with the Video and Content Platforms Research and Advanced Development Group at Cisco. Currently, he is affiliated with Ozyegin University, where he teaches and advises students in the computer science department. Ali has a PhD in electrical and computer engineering from Georgia Tech. To date, he received several academic and industry awards (including an Emmy® Award for Technology and Engineering), and was granted 30+ US patents. In 2016, he was elected distinguished lecturer by the IEEE Communications Society, and in 2018, he was re-elected for another two-year term. In 2017, he initiated and since then has been the head of delegation for the Turkish National Body for ISO/IEC JTC1/SC29 (JPEG and MPEG). He was also listed among the world's most influential scientists in the subfield of networking and telecommunications in 2020 and 2021. To learn more about Ali's projects, publications, talks, and teaching, standards and professional activities, visit https://ali.begen.net.

Roger Zimmermann (S'93-M'99-SM'07) received his M.S. and Ph.D. degrees from the University of Southern California (USC), USA, respectively. He is currently a professor with the Department of Computer Science, National University of Singapore (NUS), Singapore. He is also a lead investigator with the Grab-NUS AI Lab and from 2011–2021 he was Deputy Director with the Smart Systems Institute (SSI) at NUS. He has coauthored a book, seven patents, and more than 350 conference publications, journal articles, and book chapters in the areas of multimedia processing, networking and data analytics. He is a distinguished member of the ACM and a senior member of the IEEE. He recently was Secretary of ACM SIGSPATIAL (2014–2017), a director of the IEEE Multimedia Communications Technical Committee (MMTC) Review Board and an editorial board member of the Springer MTAP journal. He is also an associate editor with IEEE MultiMedia, ACM TOMM and IEEE OJ-COMS. More information can be found at http://www.comp.nus.edu.sg/∼rogerz.