Source Rate Control in Videoconferencing Application using State–Action–Reward–State–Action Temporal Difference Reinforcement Learning

by

Ali REZAGHOLIZADEH

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS IN INFORMATION TECHNOLOGY ENGINEERING
M.A.Sc.

MONTREAL, APRIL 27, 2022

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Stéphane Coulombe, memorandum supervisor
Department of Software and IT Engineering, École de technologie supérieure

Mr. Ghyslain Gagnon, co-supervisor
Department of Electrical Engineering, École de technologie supérieure

Mrs. Christine Tremblay, president of the board of examiners
Department of Electrical Engineering, École de technologie supérieure

Mrs. Diala Nalboulsi, member of the jury
Department of Software and IT Engineering, École de technologie supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON APRIL 20, 2022

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

This thesis has been supported by many people and I want to acknowledge their contributions.

First, I would like to express my deepest thanks to my advisors Prof. Stéphane Coulombe and Prof. Ghyslain Gagnon for their valuable guidance, mentorship, patience, encouragement, sharing their professional experience, and providing me with financial support throughout this difficult and long-lasting project. I hope they see the fruit of their mentorship and patience through the maturity of the thesis at the end. I would like to thank the Media5 Corporation and the Natural Sciences and Engineering Research Council of Canada for their financial support.

Second, I want to thank, from my heart, my wife for her full support, patience, and for taking care of my son, while she is also a Master's student in Mathematics, during such a long time to complete this project from scratch and throughout the thesis completion.

Third, I have to also deeply thank my brother, Dr. Mehdi Rezagholizadeh, and sister-in-law, Dr. Soheila Samiee, who helped me a lot with their contributions in reviewing and commenting on some parts of the thesis.

Last, but first in my heart, I thank my parents who always encouraged me to go further in knowledge. In particular, God bless my mother who has passed. I owe much of my successes and achievements to her and she will always remain in my heart.

Contrôle de débit de source en visioconférence utilisant l'apprentissage par renforcement état-action-récompense-état-action à différence temporelle

Ali REZAGHOLIZADEH

RÉSUMÉ

La proportion de la vidéo IP sur Internet a dépassé 80% et continue d'augmenter. Par conséquent, il est crucial pour les fournisseurs de services vidéo de fournir une excellente qualité vidéo aux utilisateurs finaux. De plus, Internet présente certaines caractéristiques telles que des composants hétérogènes et divers algorithmes de gestion du trafic qui ne garantissent aucune qualité de service. Ainsi, il est de la responsabilité de l'application de contrôler son débit de manière à répondre aux attentes des utilisateurs.

Plusieurs travaux ont proposé des algorithmes d'ajustement du débit d'envoi à partir du contrôle du débit source pour contrôler la taille des paquets dans le but de fournir une bonne qualité de service (QoS) ou qualité d'expérience (QoE). Quelques travaux sont appliqués et évalués dans le cadre de la transmission multimédia interactive en temps réel, c'est-à-dire pour la communication en temps réel. Ces algorithmes peuvent être classés en règles de contrôle artisanales et en contrôle automatique. Les méthodes basées sur des règles ont montré un manque de généralisation à d'autres types de réseaux motivant l'étude des méthodes automatiques. Les algorithmes de contrôle automatique existants appliqués dans les applications multimédias interactives en temps réel, telles que la visioconférence, sont basés sur l'apprentissage par renforcement (RL).

Dans ce travail, nous proposons une méthode RL tabulaire, c'est-à-dire, sur la politique État-Action-Récompense-État-Action (SARSA) en tant que méthode de différence temporelle en une étape, pour contrôler le débit source, tandis que les autres méthodes basée sur RL fonctionnant pour la communication en temps réel basée sur UDP, appliquent l'approximation de fonction (FA) comme moyen d'apprentissage du modèle. Bien que les approches RL tabulaires consomment plus d'espace mémoire pour stocker leurs paramètres, la mise à jour des paramètres n'est pas approchée comme ce qui est fait dans FA et cela peut conduire à une convergence plus rapide vers une valeur optimale. De plus, les quelques méthodes RL existantes dans un environnement de communication en temps réel formulent des récompenses en utilisant des métriques de QoS objectives qui ne peuvent qu'*estimer* la QoE qu'un utilisateur expérimente. Au meilleur de nos connaissances, nous présentons la première méthode RL tabulaire pour l'ajustement de débit pour la transmission multimédia interactive en temps réel. Nous proposons également une nouvelle perspective dans la formulation du problème de contrôle de débit en RL. Nous utilisons le PSNR, une métrique de qualité visuelle avec référence largement utilisée, pour évaluer la qualité vidéo perçue par l'utilisateur.

Dans ce travail, après avoir suggéré une manière d'appliquer et d'évaluer notre méthode dans un environnement réseau générique, nous proposons un simulateur de réseau basé sur un modèle de file d'attente classique pour nos expériences. Nous appliquons et évaluons la méthode proposée dans le contexte de la visioconférence en utilisant un codec vidéo H.264 sur l'environnement

réseau simulé. Notre méthode proposée est évaluée et comparée au réseau neuronal borné (BNN) sur deux configurations du simulateur de réseau, c'est-à-dire avec un débit binaire disponible faible et élevé. Les résultats montrent que SARSA surpasse BNN avec un PSNR, une perte de paquets et une consommation de débit significativement meilleurs.

Mots-clés: contrôle du débit source, codage adaptatif du débit, visioconférence, communication en temps réel, apprentissage par renforcement

Source Rate Control in Videoconferencing Application using State–Action–Reward–State–Action Temporal Difference Reinforcement Learning

Ali REZAGHOLIZADEH

ABSTRACT

The portion of IP video over the Internet has exceeded 80% and is still increasing. Therefore, it is crucial for video service providers to satisfy the video quality experienced by the end users. Moreover, the Internet has some features such as heterogeneous components and diverse traffic management algorithms that do not guarantee any quality of service. Thus, it is the application's responsibility to control its flow such to meet the users' expectations.

Several works proposed the sending rate adjustment algorithms from source rate control to control the packet size with the aim of providing good quality of service (QoS) or quality of experience (QoE). A few works are applied and evaluated in the context of real-time interactive multimedia transmission, i.e., for real-time communication. Such algorithms can be categorized as hand-crafted controlling rules and automatic control. The rule-based methods showed a lack of generalization to other types of networks motivating the study of automatic methods. Existing automatic control algorithms applied in real-time interactive multimedia applications, such as videoconferencing, are based on Reinforcement Learning (RL).

In this work, we propose a tabular RL method,i.e., on-policy State–Action–Reward–State–Action (SARSA) as a one-step Temporal Difference method, to control the source rate, while the other RL-based works on real-time communication apply function approximation (FA) as a way of learning the model. Although tabular RL approaches consume more memory space to store its parameters, the updating of parameters is not approximated like what is done in FA and this can lead to faster convergence to an optimum value. Moreover, the few existing RL methods in a real-time communication environment formulate reward using some objective QoS metrics which can only *estimate* the QoE that a user experiences. To the best of our knowledge, we represent the first tabular RL method as a rate adjustment control for real-time interactive multimedia transmission. We also propose a new perspective in formulating the rate control problem in RL. We use PSNR, a widely used full reference visual quality metric, to evaluate the video quality perceived by the user.

In this work, after suggesting a way to apply and evaluate a method in a general network environment, we propose a classical-queuing-model-based network simulator for our experiments. We apply and evaluate the proposed method in the videoconferencing context using an H.264 video codec over the simulated network environment. Our proposed method is evaluated and compared with Bounded Neural Network (BNN) over two configurations of the network simulator, i.e., with low and high available bitrate. The results show that SARSA outperforms BNN with significantly better PSNR, packet loss, and bitrate consumption.

Keywords: Source rate control, Rate adaptive encoding, Videoconferencing, Real-Time Communication (RTC), Reinforcement Learning

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
BNN	Bounded Neural Network
FTP	File Transfer Protocol
ML	Machine Learning
QP	Quantization Parameter
RMCAT	Real-Time Protocol Media Congestion Avoidance Techniques
RTC	Real-Time Communication
SARSA	State–Action–Reward–State–Action
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

LIST OF SYMBOLS AND UNITS OF MEASUREMENTS

$V_\pi(s)$	State-value function
$Q_\pi(s,t)$	State-action value function of a arbitrary state s and action a when following the policy π
$p\left(s^{'}, r \mid s,\ a\right)$	Dynamics of the MDP (i.e., model of the environment)
$Q_\pi(S_t, A_t)$	State-action value function of the state S_t and action A_t when following the policy π
s_{LOST}	The number of lost packets during the transfer over the network simulator formulated in state signal of proposed RL formulation
$MWT^{j}_{ts_i}$	Maximum Waiting Time considered for the j th packet belonging to the i th time slot of the trajectory
s_{LATE}	The number of packets belonging to a time slot, formulated in state signal of proposed RL formulation, that arrived at the receiver but after the maximum expected time
fps	Frame Per Second
s_{Lidx}	The index of the first packet which has not been decoded in a time slot. It is formulated in state signal of proposed RL formulation.
s_{RATE}	The sending bitrate in a time slot formulated in state signal of proposed RL formulation
BR_{min}	Minimum bitrate considered in tile coding the sending bitrate
BR_{max}	Maximum bitrate considered in tile coding the sending bitrate
N_{BR}	Number of tiling in the bitrate range
N_{QP}	Number of actions in proposed RL formulation

γ	Discounted rate in proposed RL method
PSNR_{min}	Minimum PSNR below which it is considered as unimportant to construct reward
PSNR_{max}	Maximum PSNR above which there is a small effect on constructing reward
α	Learning rate in proposed RL method
R_t	Reward value after taking action A_{t-1} in state S_{t-1} in proposed RL method
$SF(S_t, n)$	The softmax value given S_t and $n^{th} action$ in action list
$Q(S_t, n)$	The state-action value of n^{th} action given S_t
τ	Temperature in Softmax function
V_f	Target bitrate for Bounded Neural Network (BNN) method
V_e	Error between the target value and future observations in Bounded Neural Network (BNN) method
b_t	Network throughput at time slot t used in shaping the input for Bounded Neural Network (BNN) method
q_t	Delay gradient used at time slot t in shaping the input for Bounded Neural Network (BNN) method
e_t	Delay gradient on demand for time slot t used in shaping the input for Bounded Neural Network (BNN) method
$ExpT^j_{ts_i}$	The expected time for receiving the j^{th} packet in the i^{th} time slot
$MExT^j_{ts_i}$	The maximum time which the receiver waits for the j^{th} packet in the i^{th} time slot
$PLoT$	The tolerable buffer playout time

$MWT_{ts_i}^{j}$	The Maximum Waiting Time for the j^{th} packet in the i^{th} time slot
ρ_i	The bit error probability at link i
β_{ts_k}	The available bitrate value at k^{th} time slot in wireless router part of the simulator
Θ_β	The time to stay on an available bitrate value in wireless router part of the simulator
Δ	The step size used to change the bitrate in wireless router part of the simulator
Θ_Δ	The time duration to keep the selected step duration in wireless router part of the simulator
$P_{ts_i}^{j}$	The j^{th} packet of time slot i
$Delay_{ts_i}^{j}$	The delay of a coming packet, $P_{ts_i}^{j}$
$\aleph_{ts_i}^{j}$	The length of the j^{th} packet (in bits) belonging to the time slot i
$Interval^{j}$	The time interval between the arrival of the j^{th} packet, and its previous packet that arrived at that node
χ_{ts_i}	Number of all the packets arriving from other resources at the router in each time slot
μ	Router serving rate
λ_{ts_i}	The average number of arrival packets at the router at i^{th} time slot and consumed by Poisson distribution
θ_λ	The duration time which the average number of packets arriving at each time slot is kept constant
δ	The value which is added after a while to λ_{ts_i}

$\zeta_{ts_i}^{j}$	The number of packets coming from other resources before the j^{th} video packet

INTRODUCTION

The number of video contents over the Internet is increasing. Video streaming applications such as YouTube, Netflix, Hulu and Disney+ on one side and interactive real-time multimedia communication applications such as Zoom, Skype, Microsoft Teams, Google Hangout, WhatsApp, online gaming applications and many others on another side play an important role in ever increasing the multimedia contents over the Internet. Cisco presented in their annual Visual Networking Index report (latest report was in 2019, Cisco (2019)) that the IP video traffic shaped 75% of all IP traffic in 2017 and estimated to approach 82% by 2022. So, it is ever more important for the video service providers to satisfy the end user's quality expectations.

Meanwhile, although the Internet infrastructure grows to meet the traffic's demands, it only promises best effort service due to its heterogeneous components, changing topology, vast range of traffic management algorithms and routing policies, end users' unpredictable behavior (Obaidat, Zarai & Nicopolitidis, 2015), etc. all of which lead to highly dynamic network conditions (Zhang, Zhou, Lu, Ma, Hu, Li, Zhang, Ma & Chen, 2020). Thus, it is the applications' responsibility to meet their demands by adjusting their sending rate.

A wealth of rate adjustment algorithms were proposed at the application layer to fulfill such requirements from source rate control algorithms (Wu, Ci & Wang, 2007; Zhu, Zeng & Li, 2007), which try to take control of the encoder's output, to packet scheduling (Luo, Shyu & Chen, 2008) and adjusting the packet size (Jesup & Sarker, 2021; Widmer, Boutremans & Boudec, 2004). Some of these approaches design joint level control techniques (Luo *et al.*, 2008; Zhu *et al.*, 2007). Although the aim of a large portion of these algorithms is to share the network resources fairly (which are so-called congestion control algorithms), their inevitable concern still remains in providing QoS/QoE.

However, most of these techniques have been evaluated in video streaming applications which have less restrictions regarding packet delay and loss rate because there is more time for packets

to arrive without error. Such techniques can be viewed from two angles. In one aspect, these techniques are mainly categorized into three classes according to the information they consumed to estimate the network status and adjusting the sending rate accordingly: loss-based (Ha, Rhee & Xu, 2008; Hoe, 1996), delay-based (Shalunov, Hazel, Iyengar, Kuehlewind et al., 2012; Wei, Jin, Low & Hegde, 2006), and network-oriented congestion notification (Koo & Chung, 2010; Shalunov *et al.*, 2012). In the second aspect, these algorithms are categorized into rule-based methods (Hoe, 1996; Zaki, Pötsch, Chen, Subramanian & Görg, 2015), in which the estimated network status is manually mapped to actions, and automatic control (Huang, Zhang, Zhou & Sun, 2018b; Mao, Netravali & Alizadeh, 2017), which try to take an action according to the experience the application gained from previous interactions with the networks.

Many of these algorithms are only applied on top of the transmission control protocol (TCP). The algorithms which rely on some TCP mechanisms such as re-transmission and in-order packet delivery mechanisms to guarantee some level of service quality cannot be deployed in interactive real-time communication (RTC) systems like videoconferencing. The reason is that, RTC applications have high sensitivity to delay and loss and these aforementioned mechanisms in TCP protocol cannot meet the requirements of RTC applications. Some other techniques, however, such as LEDBAT (Shalunov *et al.*, 2012) and TFRC (Floyd, Handley, Padhye & Widmer, 2008) are capable to also operate on top of the user datagram protocol (UDP) and have been claimed to be of practical use in interactive multimedia transmission.

RTC applications have specific requirements. The real-time protocol media congestion avoidance techniques (RMCAT) group created by the IETF defined the requirements of interactive real-time multimedia and proposed three control algorithms in the body of WebRTC framework (a real-time protocol for browser-based applications) over RTP/RTCP protocol: Google Congestion Control (GCC (Holmer, Lundin, Carlucci, Cicco & Mascolo, 2016b)), NADA (Zhu, Pan, Ramalho & de la Cruz, 2020), and SCREAM (Johansson & Sarker, 2017). Also, there are some other works which

are not based on RTP/RTCP protocol such as Sprout (Winstein, Sivaraman & Balakrishnan, 2013), (Kurdoglu, Liu, Wang, Shi, Gu & Lyu, 2016), and Salsify (Fouladi, Emmons, Orbay, Wu, Wahby & Winstein, 2018). Such techniques suffer from generalization to a wide range of network environment (Zhang *et al.*, 2020).

Machine Learning (ML) empower the researchers to design automatic controllers to overcome this drawback of the hand-crafted methods. Bounded Neural Network (BNN, (Huang, Zhang, Zhou & Sun, 2018a)) is one of supervised learning approaches which deploys deep learning architecture to predict available bitrate in a range and control the source rate (the encoder's rate) accordingly. BNN is evaluated in video streaming environment; however, it is able to outperform GCC(Holmer, Lundin, Carlucci, Cicco & Mascolo, 2016a), TFRC(Floyd *et al.*, 2008), and Cubic methods(Ha *et al.*, 2008).

Among ML techniques, supervised learning methods need the ground truth to be able to learn their models. Finding the ground truth is sometimes difficult, if not impossible. Reinforcement learning is a way of learning by formulating the problem in a way that an agent can learn its model while interacting with its environment. So it does not need to have the ground truth to learn its model. Instead, it discovers the true action (which lead the agent to approach to its goal) by trying them.

Several RL techniques have been used and evaluated in non-interactive multimedia transmission (e.g., video streaming) such as Pensieve (Mao *et al.*, 2017) and (Huang *et al.*, 2018b). However, we could find only two works (i.e., Zhang *et al.* (2020) and Fang, Ellis, Li, Liu, Hosseinkashi, Revow, Sadovnikov, Liu, Cheng, Ashok et al. (2019)) applying RL method to control the sending bitrate in real-time communication (RTC). These two works uses function approximation way of updating the learning parameters.

In this thesis, we propose a tabular RL method with different problems formulating to control the encoder's rate by selecting appropriate quantization parameter (QP) values at the encoder (i.e., H.264). The performance of the model is compared to the BNN model after establishing a simulated conversation over two network simulators' configurations (i.e., yield low and high available bitrate)[1]. The specific contributions of our method include:

1. To the best of our knowledge, we represent the first tabular RL method in real-time interactive video transmission, in particular for videoconferencing.

2. We propose a new perspective in formulating the rate control problem in RL. We defined the environment which helped us in tabulating some effective parameters as state signals along with preventing some environment's parts to behave uncontrollably, especially the video input.

3. We implement a new classical-queuing-model-based network simulator from scratch.

The rest of the thesis is organized into three main chapters followed by a conclusion. In Chapter 2, we present, the literature review covering the conventional methods proposed for adjusting the sending rate. In Chapter 3, we provide more details on our RL formulation and the network simulator. In Chapter 4, we introduce the videoconferencing database and the network simulator configurations along with the results obtained from our experiments. We also analyze those results. We finally present our conclusions and recommendations.

[1] The source code is available via:
https://bitbucket.org/Alirghz/videoconferencing/src/master/.

CHAPTER 1

LITERATURE REVIEW

In this chapter, we present the literature relevant to transmission control over networks and then we turn to videoconferencing as a real-time transmission application. Moreover, we briefly review the solutions for improving video quality in videoconferencing.

1.1 Data Transmission over Networks

There are more than half a million diverse networks connected over the Internet (D'Alconzo, Drago, Morichetta, Mellia & Casas, 2019). Networks consist of heterogeneous components, from vast types of hardware (e.g., gates, routers, switches, links (fiber, cable), wireless access points) with differences in processing speed cycle and in the storing capacity at each node to the notable variety in the strategies taken for managing the traffic (e.g., flow scheduling, dropping policy, etc.), routing policy, load balancing, and so on. In addition, the internet has some challenges such as changes in the topology, mis-configuration (which can lead to outages) and misbehavior of some nodes. One immediate result of aforementioned challenges is in changes in routing (i.e., inter/intra-domain routing changes) which itself affects the end-to-end latency (Javed, Cunha, Choffnes, Katz-Bassett, Anderson & Krishnamurthy, 2013; Wassermann, Casas, Cuvelier & Donnet, 2017).

All of these factors, from heterogeneity of the components to the challenges in the Internet, lead to variability in network conditions and can directly affect quality of service (QoS) or quality of experience (QoE). For instance, changing inter-domain routing, itself, can increase the latency of at least 100 ms for 40% of the clients (Wassermann *et al.*, 2017).

Variations in network conditions have less impact on the user experience when a user browses web pages or bulk transfers data using the file transfer protocol (FTP). But such changes have a noticeable impact on real-time multimedia applications like video streaming, videoconferencing, and IPTV (Luo & Shyu, 2011; Van der Schaar & Chou, 2007). One way to satisfy

QoS demands is to find solutions in the network layer to classify and deliver a flow to its destination such that it meets QoS requirements. Such solutions include traffic classification (Garcia & Korhonen, 2018), traffic prediction (D'Alconzo *et al.*, 2019), packet scheduling (Ishimori, Farias, Cerqueira & Abelém, 2013), network-level congestion control (Boutaba, Salahuddin, Limam, Ayoubi, Shahriar, Estrada-Solano & Caicedo, 2018), and efficient routing (Banimelhem & Khasawneh, 2012) (Saha, Bera & Misra, 2018).

However, such provisioning is not guaranteed in today's best-effort Internet which consists of heterogeneous network infrastructure (Zhang *et al.*, 2020). Another solution is to control the flow from the user's device within the transport layer or application layer. In the following sub-section, we focus on some works done for controlling the flow to adapt to the network dynamics to provide some level of service quality (QoS or QoE).

1.1.1 Transmission Control Mechanisms

If an application sends its stream without considering the congestion on the path taken by such stream, it will affect both its own and other applications' (which are using the same path) quality of service either immediately or in a longer run.

Generally speaking, we can regulate the sending bit-rate in the end-system by applying at least one of these tasks: packet scheduling (Luo *et al.*, 2008), region-based coding (specially in multimedia application) (Gao, Dong, Zhang, Wen & Zeng, 2019; Sun, Ahmad, Li & Zhang, 2006), adjusting packet size (Jesup & Sarker, 2021; Widmer *et al.*, 2004), buffer managements (Luo *et al.*, 2008), rate adaptive encoding or source rate control (in some applications like multimedia which the original data is encoded to be transferred) (Wu *et al.*, 2007; Zhu *et al.*, 2007), or a joint design (Luo *et al.*, 2008; Zhu *et al.*, 2007). The goal of the sending bit-rate control is to provide one or several QoS/QoE specific parameter(s), such as high throughput, low delay, or high video quality. Moreover, it can aim at fair sharing of network resources (like available bitrate) with other flows on the path and preventing them from congestion. These types of methods are so-called congestion control algorithms.

There are a lot of congestion control methods in the literature which some of them at the same time can address the problem of QoS/QoE and the fairness such as Holmer *et al.* (2016a). Since the congestion controller output is usually applied as the source rate controller in multimedia transmission , we start seeing congestion control approaches being integrated in the TCP transport protocol. These techniques are evaluated in general-purpose multimedia transmission (often in video streaming). Then, we review some time-sensitive multimedia applications and their requirements to be considered in congestion control methods. Finally, we mention some methods which are evaluated in interactive real-time video communication (like videoconferencing) as very time-sensitive applications.

The TCP protocol, which is at the transport layer, provides built-in congestion control which mostly deploys the conventional TCP congestion control algorithm (Blanton, Paxson & Allman, 2009). We can categorize existing solutions in the literature into three classes based on the information type they use to infer network conditions: loss-based, delayed-based and network-oriented congestion notification.

1.1.1.1 Loss-based Approaches

Loss-based approaches control the sending rate after observing packet loss as congestion indication on the path. Generally, they gradually increase the sending bitrate (linearly or non-linearly) until a packet loss occur, then start to decrease it. TCP loss-based methods such as Cubic (Ha *et al.*, 2008), NewReno (Hoe, 1996), and RAP (Rejaie, Handley & Estrin, 1999) have been evaluated in multimedia transmission. They show several problems such as the video quality degradation because of long delay encountered after continuously increasing the bit-rate until packet loss occurs, or, because of saw-tooth oscillation of the bit-rate (Nihei, Yoshida, Kai, Kanetomo & Satoda, 2017; Zink, Künzel, Schmitt & Steinmetz, 2003).

1.1.1.2 Delay-based Approaches

In delay-based approaches, researchers take the delay information of the transmitted packets into consideration as another way of inferring congestion. The goal of delay-based approaches is to constraint the delay to converge to a specific value. But it is reported that the final delay does not converge (Geng, Zhang, Niu, Zhou & Guo, 2015) and, generally speaking, determining an optimum target value for the delay is difficult because of variations in the propagation delay among different environments (Nihei *et al.*, 2017).

Three types of delays have been considered in the literature: round-trip time (RTT) as deployed in TCP FAST (Wei *et al.*, 2006) and (Jain, 1989), one-way delay as deployed in LEDBAT (Shalunov *et al.*, 2012), and RTT gradient as deployed in CDG (Hayes & Armitage, 2011) (as mentioned in (Carlucci, De Cicco, Holmer & Mascolo, 2016)) and Verus (Zaki *et al.*, 2015) (on top of UDP but inherits the re-transmission mechanism from TCP). Deploying RTT and one-way delay to infer the congestion on the path is shown to lead to problems such as low-channel utilization in presence of loss-based flows or reverse traffic and "*latecomer effect*"[1], respectively (Carofiglio, Muscariello, Rossi & Valenti, 2010; Grieco & Mascolo, 2004). One variation of using delay gradient is to use a one-way queuing delay gradient. A case in point is Google Congestion Control (GCC) (Holmer *et al.*, 2016a) which is used in the WebRTC system on top of UDP (which will be discussed later).

Some hybrid approaches use both loss and delay signals to control the sending bit-rate such as TCP friendly rate control TFRC (Floyd, Handley, Padhye & Widmer, 2000) and TCP Vegas+ (Hasegawa, Kurata & Murata, 2000). In particular, TFRC adopts the equation-based algorithm on top of RTT, re-transmit timeout, and loss events rates to estimate the available bitrate. The sender receives at least one feedback message containing loss event rate per round-trip time from the receiver. Each time, the value of the equation is calculated and compared with the actual rate used in the last transmission and changes the bit-rate accordingly. TFRC shows slower reaction, compared to TCP, to available bitrate variations.

[1] A late flow starves the early flow when they share the same bottleneck (Carlucci *et al.*, 2016)

1.1.1.3 Network Congestion Notification-based Approaches

There are some sending rate adjustment techniques which support the network congestion notification generated by intermediate routers on the path by marking some fields in the IP header. Two examples of such network congestion notifications are Explicit Congestion Notification (ECN) (Floyd, Ramakrishnan & Black, 2001) and Pre-congestion notification (PCN) (Karagiannis, Chan, Moncaster, Menth, Eardley & Briscoe, 2012). Among these techniques, there are some works which have been employed mainly over TCP and support ECN to infer congestion in addition to packet loss or delay metrics such as (Koo & Chung, 2010), LEDBAT (Shalunov *et al.*, 2012) (supports both TCP and UDP transport layer), and TCP-LP (Kuzmanovic & Knightly, 2006).

However, the network-layer notification is not guaranteed in best-effort Internet. However, it is reported in 2012 that the UDP protocol is unable to access network-layer notification bits (i.e., ECN) by that time. It is also worth mentioning that, this network-layer notification may not be guaranteed at some nodes. In addition, in particular, ECN does not solve the issue of late reaction to congestion due to its process cycle time that takes at least one RTT delay to receive an end-to-end feedback carrying the ECN signal (Zhu, Vannithamby, Rödbro, Chen & Andersen, 2012).

The above-mentioned algorithms that have been evaluated in the context of video streaming are hand-crafted so that the observed network conditions are manually mapped to reactions. There are other state-of-the-art artificial intelligence (AI)-based rate control which deploy supervised and reinforcement learning to optimize user quality of experience (QoE) (such as Pensieve (Mao *et al.*, 2017) and (Huang *et al.*, 2018b)). As the aim of our work is to focus on interactive video communications, we refer the reader to other AI-based techniques in the survey in (Boutaba *et al.*, 2018).

Nowadays, many applications rely on TCP to deliver their contents from bulk data transfer like FTP to burst data transfer like web pages, and from non-time-sensitive to some level of time sensitivity in live or on-demand real-time video streaming. Video streaming, live

or on-demand, is mainly set to deliver its contents over TCP (Nihei *et al.*, 2017) because TCP can provide reliable transmission at the expense of some additional delay to achieve what the viewers need (e.g., video quality). However, in low-delay real-time communication (RTC) applications like videoconferencing in which the users interactively communicate, TCP connection is not practical due to its re-transmissions, in-order delivery mechanisms (Carlucci, De Cicco, Holmer & Mascolo, 2017), and additive increase multiple decrease (AIMD) sliding window control (De Cicco & Mascolo, 2010; Eggert & Fairhurst, 2008).

In the next section, we review the data transmission applications and some of their requirements.

1.1.2 Real-time Transmission Applications

Internet provides a vast range of services to customers. For example, it permits sending electronic mail, transferring files, surfing the internet, online transaction, internet marketing, downloading multimedia contents (from audio to image and video), making the voice or video calls, watching on-demand movies or TV programs over the Internet, or even to top it all off, emerging applications like online gaming and virtual reality (VR) which provide online streaming and interactive games (Herglotz, Coulombe, Vakili & Kaup, 2019). Each application has certain requirements in quality of service (QoS), which are mainly measured by delay, jitter, loss rate (or reliability), and bitrate, to provide consumer satisfaction.

Although, there are similarities in QoS requirements among multimedia-based services, there are still enough distinct requirements to branch them as a separate new category. Some multimedia contents are transferred over the network to be stored in user's devices. We can categorize the multimedia content passed over the network to three classes mainly based on their delay sensitivity. Some applications have large margins to accept delay, like live broadcasting which can tolerate up to 30 seconds of delay which cannot be perceived by the end user since the session is not interactive and there is no time reference. Another example is to download a movie. To the user, receiving all requested contents without any loss, i.e., reliability, is more important than experiencing a few minutes of delay in downloading. In contrast, interactive

audio or video transmissions like teleconferencing, Internet telephony, and gaming have tight delay sensitivity, usually below 200 ms. Jansen, Cesar, Bulterman, Stevens, Kegel & Issing (2011) report that user QoE of videoconferencing applications would be severely degraded if the end-to-end one-way delay of flows were to exceed 350 ms. In the middle of these two extreme cases we have the moderate delay sensitivity in multimedia streaming transmitted on demand, which can tolerate 1-5 seconds of delays (Van der Schaar & Chou, 2007).

Since UDP can transfer packets with shorter delays, time-sensitive applications are mainly relying on UDP to deliver their contents. But, unlike TCP, UDP doesn't have a built-in congestion control. Hence, it is the application's responsibility to control its flow so that both QoS/QoE requirements are met while respecting the other flows on the path by fairly sharing the available bitrate and network resources with them.

1.2 Videoconferencing

The demand for real-time multimedia services such as video streaming and videoconferencing grows every day with the emergence of electronic devices such as cell phones, tablets, and laptops and their ever-growing users (Van der Schaar & Chou, 2007; Zhuo, Wang, You & Xue, 2017). There are some other factors such as the COVID-19 pandemic which led to substantial needs to support work-from-home, virtual meetings/visits, virtual conferences, online medical visits (virtual Healthcare[2]) and even virtual entertainments. There are several applications such as Skype, Zoom, Google Hangout, Microsoft Teams, WhatsApp, GatherTown, and many others which support video calls or videoconferencing over the Internet. It is reported that the number of only cell phone users exceeds 4 billion users in the world (Aziz, Qureshi, Iqbal & Lestas, 2020). Considering network condition variations and its considerable consequence on multimedia QoE, it is crucial task for media service providers to provide users' quality of service.

[2] https://www.dialogue.co/en/integrated-health-platform

1.2.1 Transmission Control for Videoconferencing

For communications using real-time interactive applications, developers mainly employ UDP to deliver their contents (De Cicco & Mascolo, 2010). But UDP does not provide any rate control method by itself and it is the application layer's responsibility to control the flow. The following sub-section explore some solutions to this problem.

1.2.1.1 Rule-based Transmission Control

UDP-based applications can use some congestion control algorithms implemented in TCP as long as their data transmission mechanisms meet all of the algorithms' requirements. For example, LEDBAT (Kuehlewind, 2012) claimed that it can be used in different applications or network protocols, like those created on top of UDP. It can be inferred from some algorithms developed for interactive video transmission like SCREAM (Johansson & Sarker, 2017) that all TCP functionalities except re-transmission and in-order packet mechanisms can be compatible with the interactive communication applications. For example, acknowledgement signal, self-clocking, packet conservation principles, or window-based rate adjustment, can be deployed in the body of UDP-based congestion control part like SCREAM. Hence, one can investigate the capability of TCP algorithms to be integrated in UDP transport protocols. On the other hand, it is required to investigate the capability of a controlling algorithm to meet the application's demands. IETF created a group called RMCAT to define the requirements of interactive real-time medias. They evaluate the TFRC as a sample candidate (as Floyd *et al.* (2008) claim) to be used in real-time interactive media. The RMCAT group proposes three congestion controls (GCC (Holmer *et al.*, 2016b), NADA (Zhu *et al.*, 2020), and SCREAM (Johansson & Sarker, 2017)) in the body of WebRTC framework to work with Real Time Protocol (RTP)/RTCP (over UDP) protocol, a real-time inter-operator with different applications using web browsers over RTP (Alvestrand, 2014).

Google Congestion Control (GCC) (Holmer *et al.*, 2016a) is a combination of model-, loss-, and delay-based algorithms. An algorithm is model-based when it is trying to model the behavior of

the network or the video source like what has been worked by (Gao *et al.*, 2019). In GCC, a delay-based controller is located at the receiver. The receiver takes the benefit of the Koffman model to estimate the end-to-end one-way delay variation as a sign of future congestion. The estimated delay variation is used to determine the adaptive threshold to detect over/underuse in the buffers along the end-to-end path and finally decrease, or hold the rate (A_r). After receiving several RTCP report feedbacks from the receiver, the loss-based controller which is located at the sender adjusts its suggested bitrate (A_s) following a certain rule based on the observed loss rate. Finally, the minimum of A_r and A_s are fed to determine both the packet rate and the source encoding rate.

Network-Assisted Dynamic Adaptation (NADA) (De Cicco, Carlucci & Mascolo, 2017; Zhu *et al.*, 2020) is designed to control congestion (with adjusting video target rate and sending rate) with the help of Explicit Congestion Notification (ECN) and Pre-congestion Notification (PCN) in case where they are supported by the network in addition to the delay (estimation of the one-way queuing delay) and the loss information.

Self-clocked Rate Adaptation for Conversational Video (SCREAM) (Johansson & Sarker, 2017) controls the congestion by adjusting the congestion window with the help of delay (one-way queuing delay), packet loss rates, and ECN signals (where it is supported) to determine the upper limit to the stream size in flight. SCREAM also paces the packets with the aim of decreasing the risk of unnecessary packet losses and increased jitter caused by the effect of coalesce, i.e., when packets are sent in bursts. It is adjusting the sending bitrate in different time interval, from 20 ms to 400 ms depending on the media bitrate, that the feedback is received.

These methods are implemented in hard-coded controlling rules and such a pre-programmed rule-based methods are designed in specific network environment (Jay, Rotman, Godfrey, Schapira & Tamar, 2018) and suffer from not generalizing well under a wide range of network scenarios (Zhang *et al.*, 2020). For example, GCC collapses in wireless access technologies when TCP flows are present in the channel (Guerrero Viveros, 2019). Machine learning-based

techniques attracted a lot of attention to provide automatic decision-making and in some contexts propose potential solutions to this generalization issue (Jay *et al.*, 2018).

1.2.1.2 Artificial Intelligence in Transmission Controlling for Videoconferencing

Several works focused on applying automatic ways to control the sending bitrate in multimedia transmission by deploying a machine learning approach methods. Bounded Neural Network (BNN, (Huang *et al.*, 2018a)) is one example that deploys a neural network architecture to predict available bitrate in a range of values (instead of a singular value) and controls the encoder's output rate using that range. BNN feeds a time series of three sets: throughput parameters, delay gradients, and delay gradients on demand for the next video transmission. Its outputs are a target sending bitrate value and the error value which estimates the difference of target value with the true throughput. BNN has been evaluated in video streaming applications and outperformed GCC, TFRC, and Cubic methods.

However, few works bring such techniques, especially in reinforcement learning, in Real-Time Communication applications like videoconferencing. We found only two works, ((Zhang *et al.*, 2020) and (Fang *et al.*, 2019)), deploying reinforcement learning in the RTC domain. The first work, OnRL (Zhang *et al.*, 2020), proposed online deep reinforcement learning for real-time mobile video telephony. The learning phase is comprised of two stages. At the first stage, each user is training an individual model with its own session experiment and then after a while the models are aggregated to form a higher-level RL model at the second stage. The higher level is then connected to the first stage to help each individual model react to any unseen network dynamics. Each agent can perceive the network condition by determining the state signal as combination of packet loss rate, packet delay, delay intervals (i.e., one-way delay gradient of two consecutive RTP packets as measured in (Carlucci *et al.*, 2017)), and the receiver-side throughput. The action which an agent can take is concrete bitrate from 0,1 Mbps to 2,5 Mbps with 0,1 Mbps increments which represents the target output rate of the video encoder. The reward is shaped using four parameters of receiver-side throughput, packet loss rate and delay, and levels of video smoothness represented by throughput fluctuations. The models are trained

and used in a real-world video telephony systems, i.e. the Alibaba Tabao-live on WebRTC, to communicate over real network conditions.

In the second method (Fang *et al.*, 2019), the authors applied RL-based congestion control for the first time for a real-time communication (RTC) application like videoconferencing. They proposed R3Net and RL-based recurrent network to estimate bitrate and controlling the congestion with the aim of providing excellent QoE to the users. It senses the network conditions using the receive rate, average packet intervals, packet loss rates, and average RTT which make up the state signal. The action is then the estimated bitrate from 0 to 8 Mbps outcoming from the R3Net Actor model. The reward is shaped using three parameters of receive rate, average RTT, and packet loss rate. The model has been learned and evaluated over simulated and real-world networks.

These two aforementioned RL-based methods used in RTC environment formulate reward regardless of direct metrics representing the video quality that a viewer can perceive. Instead, they use some QoS metrics as quality of experience but objective QoS parameters can only *estimate* the QoE that a user experiences (Chen, Wu & Zhang, 2014). For example, these methods take the throughput measured at the receiver side as one of their parameters effects on the reward value. It means that more bitrate potentially can result in more reward that the agent gets. While, there has been shown that, sometimes, increasing the video bitrate cannot lead to a significantly better perceived video quality by the viewer. For instance, a video chunk encoded with a QP of 20 and a video chunk encoded with a QP of 10 are usually not significantly different visually but the second case requires a much higher bitrate to be delivered. Or, in static video scenes which significant changes in the bitrate can merely increase a small portion of video quality (as shown in (Huang *et al.*, 2018b)).

These two works used function approximation (FA) RL method to take control of the sending bitrate. The function approximation way of implementing a RL method can help the agent to determine the large or even continuous space for state and action signals. However, there is a tabular way of implementing the RL method which considers memory space for each

parameter (i.e., state-action value function). Although, in the tabular approach, the agent is able to define limited state and action space, the updating parameters are not approximated (using bootstrapping and delayed targets) like what is done in FA and this can lead to faster converge to an optimal value. In particular, Huang *et al.* (2018b); Sutton, Barto et al. (1998) claims that tabular RL methods like state–action–reward–state–action (SARSA) and Q-learning in Temporal Difference (TD) are effective in solving the problems with small state space.

In our work, we propose a tabular on-policy SARSA-based one-step TD learning source rate controller and evaluate its performance in videoconferencing context using an H.264 video codec over a simulated network environment. The model formulates state signals using four parameters: first, the number of packets considered to be lost because of link errors or heavy congestion on the path; second, the number of packets that exceed their tolerable arrival time to be decoded on time; third, the index of first lost packet (as the decoder is unable to decode the other packets till it receives the next Intra frame); fourth, the sending bitrate. The action signal determines the QP parameter of the encoder and gets values among six QPs (20, 25, ..., 45). The reward signal is shaped by a function of Full-Reference QoE metric (PSNR) which is calculating actual difference between the deteriorated video and the original as a representation of viewer-side perceived video quality.

In the next chapter we bring more details on our formulation of the problem using RL, our proposed method, the videoconferencing system overview, and the network simulation part. Then, in the following chapter, we evaluate our method and BNN (as the baseline) to transmit videoconferencing contents and to compare them.

CHAPTER 2

METHODOLOGY AND PROPOSED METHOD

In this chapter, we introduce our reinforcement learning (RL) formulation of the problem and selected the RL method for controlling the source rate. Then, we discuss how the model is going to be learned and evaluated in this work. Finally, we review the system and the classical-queuing-model-based network simulator we implement.

2.1 Source Rate Control with Reinforcement Learning

In this subsection, we describe how we formulate the videoconferencing source rate control problem and we introduce an RL method we used in the experiments (i.e., tabular SARSA one-step TD). But before that, we start by introducing a few concepts as background which is required to understand how the proposed method is going to learn to solve the problem at hand.

2.1.1 Background on RL SARSA Model

Machine Learning (ML) techniques are often categorized into supervised, unsupervised, and reinforcement learning (Muhammad & Yan, 2015). Supervised ML learns its model according to a pre-existing ground truth. However, in many problems, finding such ground truth is difficult and requires a lot of human effort, if not impossible. For example, in the context of rate controlling, it would need human supervision to determine the correct bitrate according to recently observed network conditions which is impossible to find in the real-world environment. Unsupervised learning, on another hand, does not need to have pre-prepared labels but is practical in the problems which need to cluster data based on their similarities.

Reinforcement Learning is designed to learn what to do by translating experienced situations to the action over interacting with an environment. So, there is no existing ground truth indicating the agent (the learner) which action to take, but instead it must discover the optimum action, which leads to maximum expected value of the reward, by trying them (i.e., trial and error).

Another feature that distinguishes the RL from other ML types is that it can be applied in the sequential decision-making problems, like in our rate controller problem, where the selected action not only affects the immediate reward achieved but impacts the upcoming situations and, accordingly, the future rewards.

RL problems can be mainly formulated in such a way that the learner can learn from interaction with the environment to achieve a goal using finite Markov Decision Process (MDP), which is an idea from dynamic system theory (Sutton & Barto, 2018). MDP is a formalization of sequential decision-making in which the taken decision not only affects on the immediate reward but the future situations and rewards. In this regard, any problem can be formulated using RL when it fulfills these two conditions: i) involves interacting with an environment in a sequential decision-making manner, ii) reward can be determined such that maximizing the expected return over time (i.e., cumulative sum of a (discounted) received reward (as a scalar value) as in section 3.3 of (Sutton & Barto, 2018)) leads to achieve the goal of the problem. The agent in this formulation can learn from interacting with the environment and approach its goal (Sutton & Barto, 2018).

The framing of an RL solution to a problem consists of: i) a learner which is called the *agent*, ii) the phenomenon outside of the agent with which it interacts is called the *environment*. The interaction between the agent and the environment is formulated by i) the *state* that the agent senses from the environment, ii) the *action* on which the agent makes decision, and iii) the *reward* which the environment gives to the agent because of taking that action on the state. The agent receives a reward feedback and sense its new state from the environment. These interactions are illustrated in Figure 2.1. A sample sequential decision-making is brought in backup diagram in Figure 2.2, which we now explain. Consider we observe state s_t after interaction with environment. We have several options to choose as an action but we take one which the policy that has been learnt suggests (i.e., a_t). Consequently, this decision leads to achieve and sense specific reward r_{t+1} and state s_{t+1} after interacting with the environment. This cycle of observing a state and taking an action according to the policy and reaching to a

new reward and state is continuing until the end of the determined interactions (i.e., episode or trajectory).

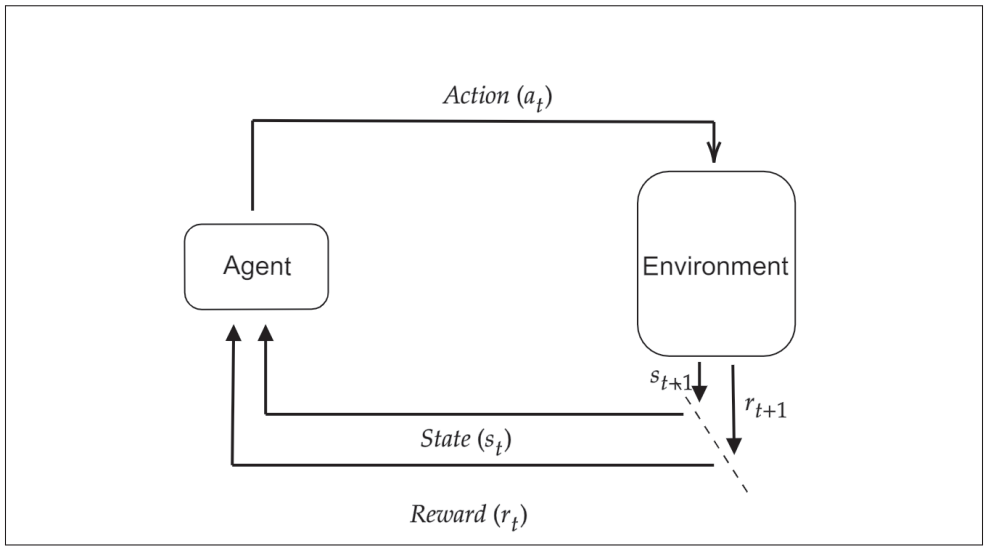


Figure 2.1 The agent-environment interaction in MDP

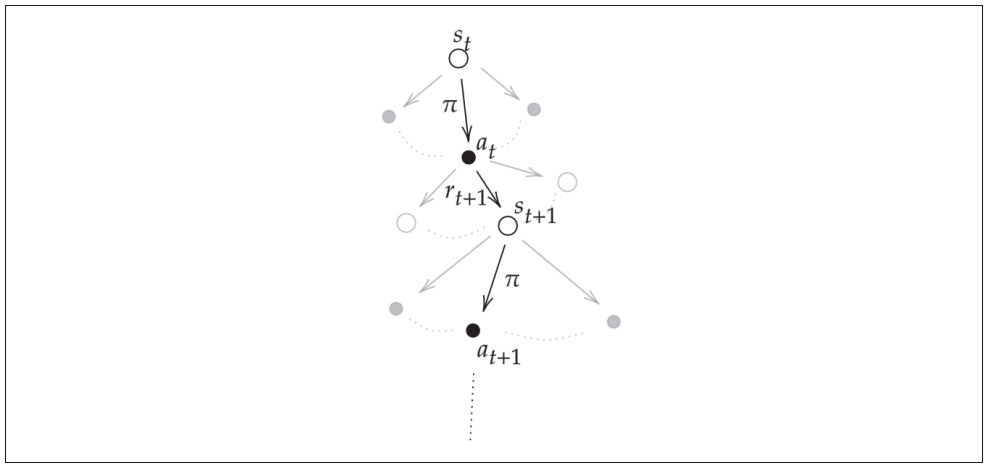


Figure 2.2 A general backup diagram rooted at a state followed by a sample sequential decision-making. White circles represent states and black circles represent actions. Arrows from an action to a state represent the reward that is achieved after interaction of the agent with the environment

In RL, a *policy* is a translation of state to probabilities of choosing each action. The goal of an RL task is to find a policy state that gives more rewards over the long run. To achieve this,

almost all RL algorithms consider a *state-value function*, $V_\pi(s)$, (or *state-action value function*, $Q_\pi(s,a)$). Given a policy, they represent how prospective it is for the agent to achieve more overall rewards if the agent is in a given state (or how prospective it is if the agent takes a given action in a given state). At the beginning, these parameters have zero values (or for the policy, it starts from uniform action selection). Over the time, the policy and the value functions are iteratively adjusted toward optimality (or approximate optimum). Following a policy, the value functions are updated and then, given a value function, the policy is updated. A better than or equal to a policy is one that leads to higher or equal value function.

The RL base models are varied based on how a model is going to update aforementioned parameters. Dynamic programming (DP) is updating a value function of a state based on all possible actions, states and rewards it may encounter in one-step ahead planning over simulated experience. Figure 2.3 presents how the DP updates the value function on the basis of other estimates (bootstrapping) related to those states, actions, and rewards succeeding the current state using the complete model of the environment (i.e., environment's dynamics). So, DP assumes that a perfect model of the environment, $p\left(s^{'}, r \mid s,\ a\right)$, is available.

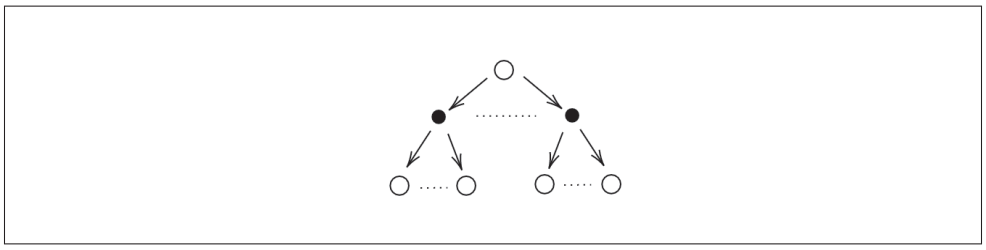


Figure 2.3 Dynamic Programming backup diagram

In many problems, the prior knowledge of the environment's model is not available. Monte Carlo (MC) methods (Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis & Colton, 2012) compensate for this inability by falling themselves in *actual experience*. Learning is done by updating the value function using the average sample returns (of rewards) after reaching the end of a trajectory, as illustrated in Figure 2.4. Another learning method in RL is Temporal-Difference (TD) learning which has the benefits of both MC and

DP. TD learning (Sutton, 1988) is like Monte Carlo in a sense that it learns directly from actual experience without requiring the model of the environment. TD is like dynamic programming in a sense that it updates the value functions based on other currently estimated values (it bootstraps) without waiting to reach the end of a trajectory to get the real final outcome (Sutton & Barto, 2018). If the TD uses n-step ahead value functions to update a given state or an action in a state, it is called n-step TD.

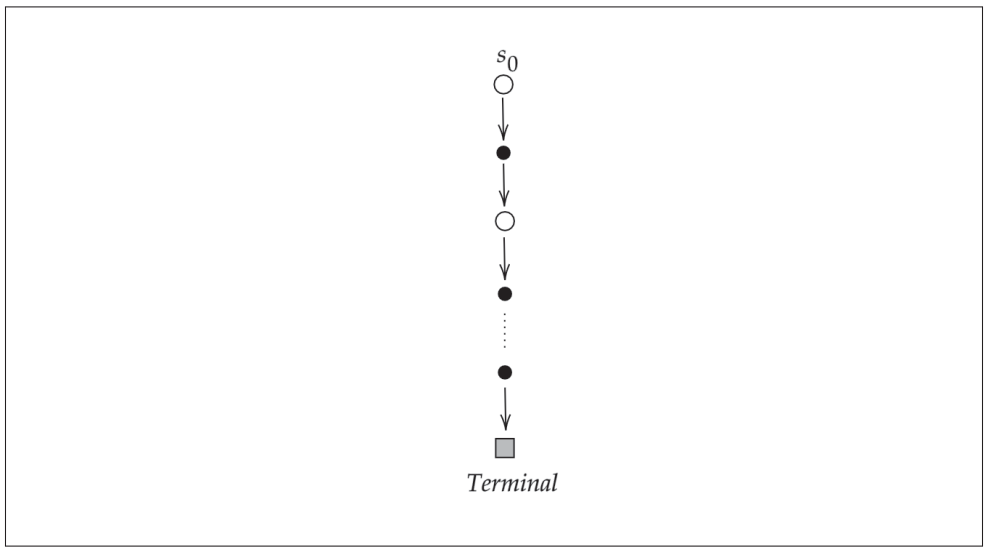


Figure 2.4 Monte Carlo backup diagram. S_0 is the initial state that the agent senses from the environment in a trajectory. The agent interact with the environment by selecting an action until it reaches to a end time or terminal state. The updates are performed after a trajectory is completed

SARSA and Q-learning are two examples of TD method. In one-step TD, they update state-action value of a given state and action respectively using the state-action value of the upcoming action or the maximum state-action belonging to the next state, as illustrated in Figure 2.5. More details are provided in sections 6.4 and 6.5 of Sutton & Barto (2018).

One challenge in RL is how to manage exploring all actions along with exploiting what has been learned so far in the policy. This challenge is important in a sense that it can affect the convergence of the agent's parameters. On-policy is the term used for the methods which try to use currently learned policy to explore, and off-policy is a term for a way that is not going

Figure 2.5 Backup diagram of SARSA and Q-learning. a) SARSA uses the state-action value of the upcoming action to update the state-action value. b) On the other hand, Q-learning uses an action of the next state which has maximum state-action value

to involve the learned policy but instead takes different strategies for exploration. SARSA is on-policy and the Q-learning is off-policy method.

The aforementioned algorithms can be used in tabular or function approximation format. In *tabular*, the value function is represented as a table and the algorithm tries to calculate the true value of the value function (following a policy). But, in *function approximation*, an approximation of a true value of value function (following a policy) is calculated using a weight vector. So, instead of storing a table, it needs just to store the weight vector which maps states (or state-action signal) to approximate value function. As the RL methods in tabular format needs to reserve memory for all state or state-action space, there is a limitation with applying them to large state and action spaces. Function approximation, on another hand, enables to handle an RL problem with large or continuous state action spaces (Xu, Zuo & Huang, 2014).

Despite the memory limitations with the tabular format, in this project, a method (i.e., SARSA) in tabular format is used. This is motivated by two factors: I) as mentioned above, the methods in tabular format calculate the true value of the value function and reaching to the convergence is more probable; and II) the state signal is found such that it is able to cover several main factors affecting the video quality experienced by the viewer. The main factors are obtained with the help of different viewing environments to cover all parts outside of the agent (not only

the network) affecting the goal, such as input video, encoder, decoder, etc. So the state space does not need to be large.

2.1.2 Formulating RL for Source-rate Control in Videoconferencing Systems

The goal that we intend to achieve in the source rate controller of the videoconferencing system is to obtain as much quality of experience as possible considering what is allowed by the network conditions. Generally speaking, RL is a suitable approach to achieve this goal because of a main reason: RL tries to increase its cumulative discounted long-term rewards. So, it is flexible-enough to miss some immediate high rewards with the hope of compensating the losses in future decisions. This is what is needed when dealing with the difficulties encountered in multimedia transmission over wireless networks. Indeed, selecting the highest video quality at a given time may cause future congestion. Therefore, the strategy must consider the long-term video quality and not only focus on the current frame. However, to enable an RL method to meet to this objective, it is vital to carefully and precisely design the problem in RL.

We formulate the problem in RL by determining Agent, Environment, State, Action, and Reward signals such that they can meet this target. As mentioned in the previous section, there are several Temporal Difference RL methods such as Q-learning and SARSA. However, there is no pre-defined rule to state that one method is better than other methods in a problem, except than trying them. In this work, we select the tabular one-step SARSA method as a RL method to update the learning parameters (i.e., state-action value function, $Q_\pi(S_t, A_t)$. Upon learned parameters, the policy maps a state, which the agent senses from its environment, to proper action with the aim of approaching to the determined target; obtaining high QoE (i.e., PSNR).

2.1.2.1 Agent and Environment

The agent is the learner in RL which selects an action according to the state of the environment it is observing. The environment is all things outside of the agent. In our work, the agent is the core which makes the decisions about the Quantization Parameter (QP) of the video encoder.

In our work, we consider the H.264 video compression standard (Wiegand, 2003). In video compression, the QP controls the compromise between video quality and bitrate. The lower QPs lead to a higher quality at the expense of higher bitrate and higher QPs lead to lower quality and bit rate. It is important to note that for a given QP value, the resulting quality and bit rate can vary greatly depending on the actual video content. For instance, at a given QP value, video sequences comprising complex spatial textures and motions will lead to higher bitrate compared to those with uniform and static spatial content. The environment is considered as a combination of input video, the video encoder, the network simulator, the video decoder, the rules that governs the application (such as the mechanisms applied for fetching the received packets, the jitter buffer, etc.), all of which can affect the state signal, the perceived QoE by the viewer, and the reward.

We determine the agent-environment interaction as *continuing task* but it is spread through trajectories. Each trajectory contains several warm-up time slots and specific communication time slot (i.e., the same as the length of video input). The video system considered divides the video information into time slots. Each time slot comprises a group of pictures (GOP)[1] starting with a first intra frame, coded independently, followed by a number of inter frame, each predictively coded based on the frame that precedes it. Thus, if a packet containing a frame is lost, all the following frames in the time slot it belongs to cannot be reconstructed and are considered lost. For instance, if the intra frame is lost then all the frames of the time slot are lost. In each time slot, the agent must determine the best QP value. In our experiments, we consider time slots of 1 second. The warm-up phase is required especially for the baseline method (i.e., BNN that we used to compare our method againts) because it needs to have initial conditions to operate. As a matter of fact, the baseline method (i.e., BNN) requires information from the last 8 time slots to operate. On the other hand, our method (i.e., based on SARSA) can operate from a single previous time slot.

[1] See https://www.haivision.com/resources/streaming-video-definitions/group-of-pictures/#:~:text=With%20the%20notable%20exception%20of,bitrates%20can%20be%20significantly%20reduced.

We position the agent on the sender side rather than on the receiver side because the encoder is on the sender side and one state signal is provided from the sender side (i.e., the sending bitrate). Thus, when the agent is located on the sender side, the system needs a single feedback signal from receiver to the sender. But, if the agent would be located on the receiver side, it would need to have two feedback signals, one from sender to receiver and another from the receiver to the sender.

2.1.2.2 State

The state represents the network conditions perceived by the receiver and the application units' performance. It contains four parameters. First, we have for each time slot the number of lost packets during the transfer over the network simulator, denoted by s_{LOST}. This type of packet loss is recognized if packet j has not arrived before the timeout $MWT^{j}_{ts_i}$ for time slot ts_i. Second, we have s_{LATE}, which represents the number of packets exceeding another timeout, i.e. maximum expectation time denoted by $MExT^{j}_{ts_i}$ for the j^{th} packet arrive in time slot ts_i. So, these two types of packets are considered as lost packets and will not participate in generating the output video at the receiver. Third, we have s_{Lidx} is the index of the first packet which has not been decoded in that time slot because it was lost or did not arrive in time. The reason for merely considering the index of the first packet that was not decoded is that the rest of the packets arriving afterwards are not playing any role in decoding.

The last parameter, s_{RATE}, used in the state is the sending bitrate in that time slot. The bitrate parameter is a continuous variable, so it takes a lot of memory if the state is being applied within a tabular reinforcement learning methods like SARSA. Therefore, we need to convert it to a discrete form. We used the one 1-D tiling approach with hand-selected $N_{BR} = 19$ bitrate values to cover the range of BR_{min}=8 kbps to BR_{max}=3360 Kbps to represent this continuous variable. The first three parameters mentioned above are packed at the receiver and transferred back using the feedback signal to the sender. The fourth one would be added at the sender side. The agent which is located at the sender, gathers them to take action accordingly.

The state space can affect the finding of the optimal policy and the speed of the convergence to an optimal value function in RL problems (Sutton & Barto, 2018). Usually, there is non-equal chance to naturally visit (over the interaction) parts of the state space and some parts might be rarely experienced. This issue can be more severe for the problems with large state spaces, especially for the tabular RL methods. In the case of such large space, function approximation RL methods are able to generalize well, i.e., to make sensible decision for new states observed using those states that are in some sense similar and encountered before (Sutton & Barto, 2018). Considering tabular RL methods, another concern with large state space is the memory needed for keeping each state.

As the SARSA technique is a tabular RL method which consumes considerable memory, it is worthwhile to check the space of the state determined. During the experiment, the frame rate expressed in frames per seconds (*fps*) was set to *fps*=30. Therefore, because the time slots are of 1 s, s_{LOST} can take *fps* + 1=31 possible values from 0 to *fps*=30 for each time slot, i.e. from zero lost packets to all being lost. Depending on the nature of the network, most of its values may not be observed. s_{LATE} also takes *fps* + 1=31 possible values from 0 to *fps*=30, but it has a correlation with s_{LOST}, i.e., the possible values for the second parameter depend on the first parameter. Let the number of lost packets in the network (e.g., due to the bit-error) be l, then the second parameter can range from 0 to $fps - l = 30 - l$. Some values of this range, though, might be rarely experienced, depending on the network. The third parameter also takes *fps* + 1=31 values from 0 to $fps - 1 = 30 - 1$ and value *30* representing no frame loss. Lastly, the fourth parameter, i.e., sending bitrate, can take N_{BR}=19 values. Therefore, the space would be $(fps + 2) \times (fps + 1)/2 \times (fps + 1) \times N_{BR} = (32 \times 31)/2 \times 31 \times 19 = 292\,144$.

Depending on the network conditions, some parts of the state space might be observed more frequently than others and some parts may not be observed at all. To reach a convergence in policy, one factor is to visit as much as possible all probable states by the agent during its learning phase.

Still, to observe enough of the state space during the experiment, it should contain a good portion of the trajectories in the training phase. Our experiments will consist of 20 000 training trajectories over a randomly established network simulator. Each trajectory will contain a 10-time slot conversation; hence it will be possible to observe 2 000 000 states. Moreover, depending on the network's behavior during the experiment, it is possible that some parts of the state space will not be observed. Nevertheless, 20 000 training trajectories is expected to be enough to reach a convergence in state action value, as long as we select properly the other factors that are playing a role in the convergence of a RL method (specially SARSA), e.g., learning rate, discount value, how we determine the reward, etc. Convergence curves will be analyzed to ensure that this assumption is verified.

2.1.2.3 Action

The agent interacts with the environment by taking an action. The environment moves the agent from a state to the same or to another state, in response to the action. The agent in this work takes control of the Quantization Parameter (QP) of the H.264 video encoder. We determine N_{QP}=6 discrete QPs to be selected as the action signals: 20, 25, 30, 35, 40, and 45. A QP of 20 leads to the highest video quality (subsequently higher video encoded bitrate) for a video chunk compared to other QPs. As the QP increases, the video quality and hence the bitrate decreases.

So, a higher bitrate in encoded stream is translated into higher encoded video quality, but has more chance to be subject to more delays and losses after being transmitted over the network. Such delays and losses can, however, impact negatively the video quality for the viewers. On the other hand, the smaller the encoded bitrate the smaller the chances of experiencing losses or large transmission delays, but this comes at the expense of lower video quality for the viewers when the video stream is received in time and without losses. So there is a trade-off between sending high-quality video that may experience delays and losses and low-quality video that is likely to be received without impairments. The RL agent intends to take an action during a conversation providing as much quality of experience as possible that the network conditions allow; i.e. sending the highest video quality that the network conditions allow without impairment.

2.1.2.4 Reward

Determining the reward signal is a crucial part of the RL formulation in a sense that it is the only way that we can truly indicate *what* we want the controller to achieve (Sutton & Barto, 2018). The rewards should be determined in a way that maximizing the *expected (discounted) return* in a state-action value function leading to approaching to the main goal. The reason is that the agent's goal is to make a decision at each time such that it maximizes the cumulative (discounted) long-term reward.

In this work, we used the expected discounted returns to be counted in a state and action pair in value function:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.1}$$

$$Q(s, a) = E[G_t \mid S_t = s,\ A_t = a] \tag{2.2}$$

where γ is the *discounted rate* and generally can get value of $0 \leqslant \gamma \leqslant 1$. If $\gamma = 0$, the immediate reward is only considered in the gain equation and if $\gamma = 1$, then, all future rewards are counted in the equation and with more strength. If $0 < \gamma < 1$, the gain function in Eq. (2.1) is constructed by all future rewards but with reducing the strength of the reward in calculating the gain as the time goes.

As the interaction between agent and environment is determined as continuing tasks (which needs infinite sum in Eq. (2.1)), it is required to determine γ and reward such that the gain in Eq. (2.1) has finite value. If $\gamma < 1$, the gain value is guaranteed to be of finite value as long as the reward sequence is bounded.

In our work, the reward sequence is bounded by the interval (0, 1) according to the average of peak signal-to-noise ratio (PSNR) values which is defined as:

$$PSNR = 10 \log_{10} \left(\frac{255^2}{MSE} \right) \tag{2.3}$$

where *MSE* is the mean squared error between the original video, before compression, and the decoded one.

At each time slot, the PSNR of each decoded frame is calculated. The average of these PSNRs in a time slot is used to formulate the reward, Eq. (2.4). The reward after seeing the average PSNR of PSNR_{min}=30 dB or below is assigned to (0, 0.1), and otherwise, to the (0.1, 1) range using a *tanh* function as illustrated in Figure 2.6. A considerable part of the reward (around 0.8 of the reward span) is assigned to the PSNR range of PSNR_{min}=30 dB to PSNR_{max}=40 dB, which is the zone where the quality increases significantly with increasing PSNR values. Below PSNR_{min}, the quality is considered unacceptable and above PSNR_{max} it is considered excellent. There is no significant visual differences for videos having PSNR values above PSNR_{max}=40 dB and the increase in bitrate to achieve higher PSNR values is not justified.

$$Reward = \begin{cases} \tanh\left(\frac{PSNR - PSNR_{min}+0.803}{8}\right) & , \text{if } PSNR > PSNR_{min} \\ \left[\tanh\left(\frac{28\times(PSNR - PSNR_{min})}{30\times11}\right) + 1\right]/10 & , \text{otherwise} \end{cases} \tag{2.4}$$

2.1.3 Proposed One-step On-policy SARSA Method

We selected on-policy SARSA in this work which is one-step TD learning method to update the state-action value function with the aim of approaching to the expected return for that state and action, as mentioned in Eq. (2.2).

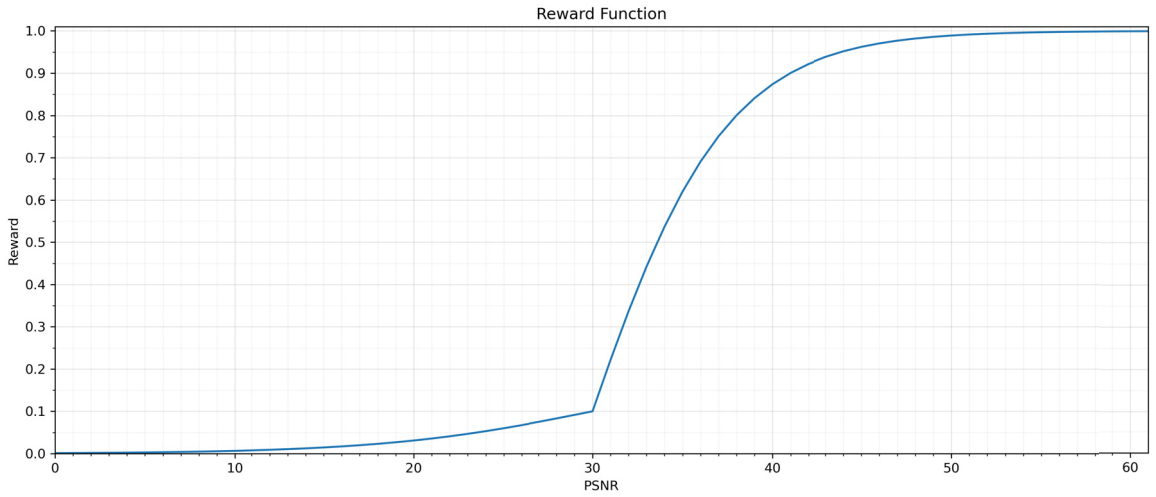


Figure 2.6 The function applied for formulating reward upon the PSNR value as QoE metrics

While training, the state-action value function, $Q_\pi(S_t,\ A_t)$, is updated by bootstrapping using the value function of the next state and action combination, $Q(S_{t+1},\ A_{t+1})$, as brought in Eq. (2.5). We have:

$$Q_\pi(S_t,\ A_t) \leftarrow Q_\pi(S_t,\ A_t) + \alpha \left[R_{t+1} + \gamma Q_\pi(S_{t+1},\ A_{t+1}) - Q_\pi(S_t,\ A_t)\right] \tag{2.5}$$

where, R_{t+1} is the reward given to the agent after taking the action, A_t, at the state S_t. α is learning rate and γ is the *discount rate* to determine the worth of future rewards to be counted in the calculation of the current return value (i.e., discounted cumulative reward). In the experiments, the learning rate α is set to 0.5 and the discount γ is set to 0.9.

We used the Softmax function with temperature parameter as the policy to map a state to the actions using values of Q for each action given the state. Softmax has been used widely in RL as a strategy to select action in each state (Collins, Brown, Gold, Waltz & Frank, 2014; Khamassi, Velentzas, Tsitsimis & Tzafestas, 2017; Vamplew, Dazeley & Foale, 2017) . Softmax can be described through the following equation (Tokic, 2010):

$$SF(S_t, n) = \frac{\exp\left(\frac{Q(S_t,n)}{\tau}\right)}{\sum_{k=1}^{N} \exp\left(\frac{Q(S_t,K)}{\tau}\right)} \tag{2.6}$$

where $Q(S_t, n)$ denotes the state-action value of n^{th} action (in action list) given S_t.

Equation(2.6) determines the extent of which we let the agent to randomly select other actions in each state (i.e., exploration) than what is currently estimated to be best (i.e., exploitation). An action with a larger state-action value receives more chance to be explored but those with lower values still have chance to be selected. The temperature parameter has an important role in this exploration/exploitation trade-off (He, Zhang, Ao & Huang, 2018). Higher temperature is meant to let the agent explore more (Tokic, 2010). After several trials, we choose a temperature value of 2 for our experiments.

2.1.4 Training the Models

To train the SARSA, 20 000 trajectories of videoconferencing conversations were established. The video calls are limited to the length of the videoconferencing clips that have been used, i.e., 10 seconds or time slots. Each trajectory in the training phase randomly selects a video clip from the database and passes its video chunks over a network simulator with new and randomly selected parameters. A trajectory intends to simulate a conversation over a fixed source and destination located in two different subnets in the Internet as shown in Figure 2.11. In the experiments, the learning rate, discount, and temperature have been selected after trial and error and finally were set to 0.5, 0.9, and 2, respectively. So, they could be further optimized. Some details on these parameters were discussed in 2.1.3.

To accelerate the experiment, we do the following two tasks. First, the experiment uses two types of pre-stored files: (i) the encoded stream of the video chunks with all actions (QP values) and (ii) all possible QoE values after decoding the stream that arrived at the receiver (considering all possible indexes of the first packet that can be lost). Second, the trajectories are managed to concurrently run over all CPU cores, which resulted in completing the entire experiments

in approximately one day. The hardware we used was an AMD Ryzen Threadripper 1950X 16-Core Processor (32 CPUs). To update SARSA's state-action parameters without blocking each other while the multiple concurrent tasks are running, tasks are allowed to update and use those action values that do not belong to the same state. It means that the tasks are not able to override the action values of the states which are already being driven by another task.

Training trajectories are split into 100 segments. Each split contains 200 trajectories. Based on the number of cores, a segment is managed to be carried out by several batches of concurrent trajectories. The split helps us track the performance of the SARSA throughout the experiment (e.g., to monitor its convergence).

We compare our system against BNN. The BNN's model architecture is shown in Figure 2.7. What the BNN needs for its offline learning is the log data experienced during conversations. As the BNN model that we use for our experiments has not been trained with the available database by (Huang *et al.*, 2018a), we use the log data of the SARSA gathered during its training phase to feed the BNN. Another reason to use the SARSA log data to train the BNN is to let the BNN being trained under the same network condition as the SARSA for fairness. The input data of the BNN is then constructed to form lists of the time series vector containing: throughput (b_t), delay gradient (q_t) and delay gradient on demand (e_{t+1}) to be experienced in the next time slot. The output of BNN consists of two values: target bit-rate V_f and the error value of V_e (error between the target value and future observations). BNN determines $[V_f - V_e, V_f + V_e]$ as a range for predicted bit-rate.

In summary, as explained the training phase of the main SARSA is using a different approach from BNN. The SARSA is trained within the conversations, while the BNN is trained offline using the data extracted from the terminated conversations, say, trajectories. It is also worth to mention that the systems and environment are written in Python and the main Python libraries we used are: Numpy, multiprocessing, threading, subprocess, random, bz2, pickle, re, pandas, matplotlib, and seaborn.

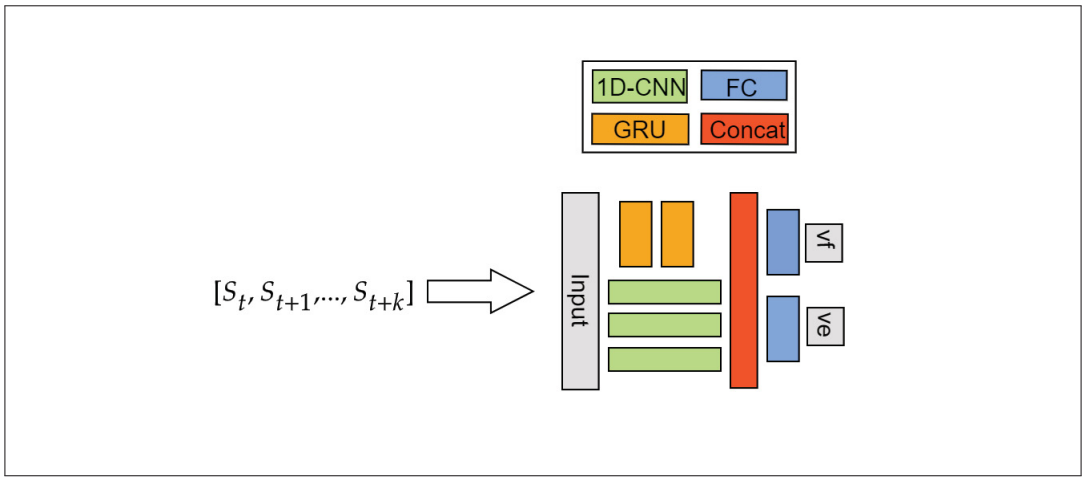


Figure 2.7 BNN model architecture. BNN gets historical data of [s_t, s_{t+1}, .., s_{t+k}] and estimates both target value for the bitrate and its error between the target value and the next bitrate observed. In this experiment, k is set to 8 and s_t assigned to (b_t, q_t, e_{t+1}), which are throughput and delay gradient of time slot t, and delay gradient on demand for time slot $t + 1$ respectively

2.1.5 Evaluating the Models

A test trajectory starts with k warm-up time slots needed to initialize BNN and SARSA methods. During the warm-up, the first second of the videoconferencing clip is chosen to be encoded in fixed low bit-rate and to be transferred over the network simulator. The last time slot of the signaling prepares the first state for the SARSA to start the conversation. At each time slot, SARSA uses greedy action selection according to the policy that has been learned to pick a QP. BNN, on the other hand, uses all k time slots of warm-up duration to shape its $k = 8$ tuples as the input vector at the first time slot of the conversation.

Then, in each time slot of the conversation, a new tuple is added to the input vector and the oldest one is discarded. Based on the output of BNN's neural-network-based model, i.e., predicted range for the available throughput, the appropriate QP is selected such that its encoded stream's length falls into the predicted range which is closer to the target bit-rate. If not found, the stream with nearest size to the lower bound of the predicted range is selected.

The models are evaluated after running over 1000 test trajectories and two configurations of network simulator (to generate two network conditions of low and high available bitrate) at the end of learning phase. The performance metrics considered to compare the methods are the average number of packets that exceed the timeouts in a timeslot, and the average PSNR for each group of pictures. These values are averaged over all time slots of all tested trajectories.

2.2 System Overview

The videoconferencing system we used consists of a sender and a receiver as shown in Figure 2.8. The sender picks a video and reads a one-second video chunk at each time slot of the conversation to be encoded based on the agent's decision. The encoded stream is then divided into packets before transmitting over the simulated network. Each packet represents an encoded video frame. In this project, no further fragmentation is performed on the packets, i.e. each packet contains a video frame. The network simulator then does its best effort to deliver the packets to the receiver. The receiver, at the end, decodes the packets that arrived on time, i.e., arrived no later than an acceptable delay, and plays them back on the user's screen after the packets passed an arrival filter unit.

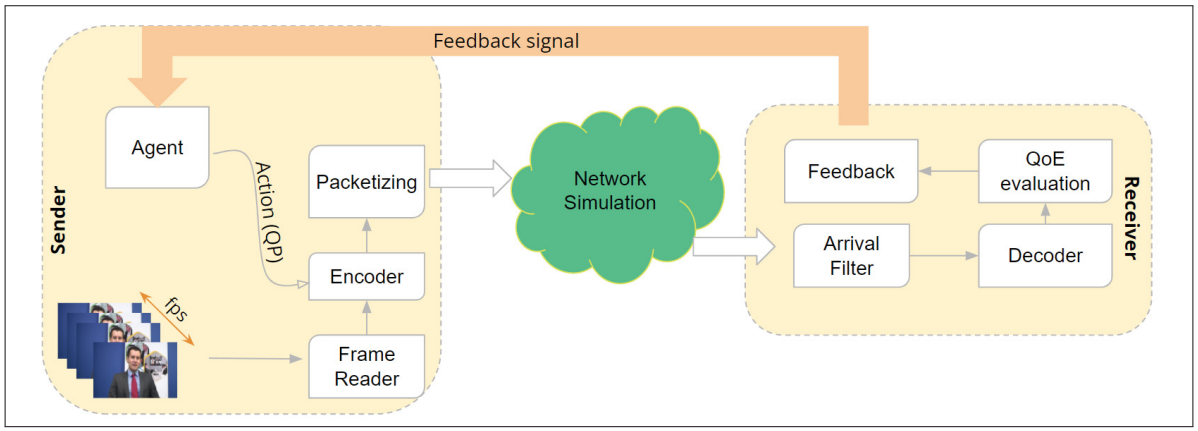


Figure 2.8 Components of the videoconferencing framework in one time slot at which the system transmits a real-time videoconferencing contents over a network simulator

Finally, the receiver gathers and send back the statistics as a feedback. The feedback signal, which contains the information required for the sender to construct its reward and the state perceived from agent-environment interaction, is fed to the agent at the sender side to enrich its experience and to take action accordingly.

As mentioned, before a simulated video conversation is established, the sender uses the first time slot of the video chunk for warm-up part of a trajectory. One time slot duration for warm-up is enough to synchronize both ends and to provide the initial state for the SARSA method, but, regarding the baseline data inputs, $k = 8$ time slots of warm-up was set in the application as illustrated in Figure 2.9. During the warm-up, both methods behave and are treated in a similar way, hence, reducing the effect of bias in the test phase. The details of the application's units are presented in the followings subsections.

2.2.1 Arrival Filter

Due to the time unit of the application being a time slot set to one second, the arrival filter reads one-second video chunks from a video file. In this work, the application reads stored video files instead of capturing from the camera, for three reasons. First, it makes it easier for the video to be used as a reference to calculate the Full Reference QoE, i.e., PSNR, after decoding of the stream at the receiver side. Second, it speeds up the experiments by providing and reusing the different encoded streams of a video chunk because encoders averagely take considerable time to encode a chunk. Third, researchers have more capability to analyze the conversation and replicate experiments to provide fair comparison between various methods if the experiment uses the stored videos.

2.2.2 Encoder Unit

A time slot of the raw video needs to be compressed to be able to be transmitted efficiently over the network. To compress a video chunk, the Encoder unit deploys libx264 video encoder inside

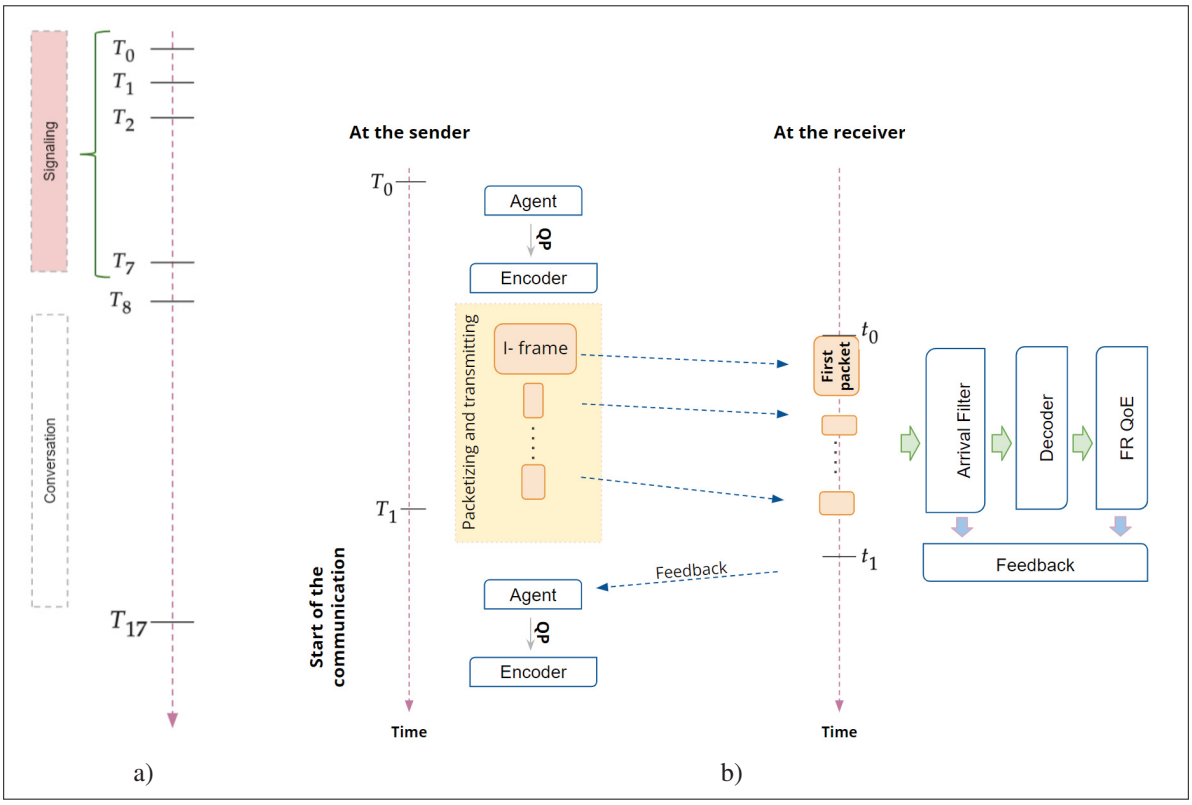


Figure 2.9 Overview of a trajectory. a) $k = 8$ time slots are set for the warm-up. During the warm-up, the first second of the videoconferencing clip is applied for transmission. Then, the conversation is being established till the end of a clip. Clips used in this project have 10-second length. b) shows the first time slot of a conversation. The first packet arrival time is considered as the base time at the receiver

the FFmpeg software[2]. The software is publicly available and it contains essential features, e.g., include CAVLC/CABAC entropy coding, 8x8 and 4x4 adaptive spatial transform, interlacing (MBAFF), lossless mode, psy optimizations, etc. The detailed options that FFmpeg provides for the libx264 can be found on its documentation (makeinfo, 2021; Shikari).

Among these options, we chose the Quantization Parameter QP to be controlled by the agent. Once the QP parameter of the encoder is set for a frame, the encoder compresses every macroblock in a frame according to the quantization value. QP value ranges from 0 to 51 in H.264, as mentioned in (Robitza, 2017). The QP of zero would do compression in lossless way.

[2] https://ffmpeg.org/

Increasing the QP value leads to degrading the quality of the video and saving more bitrate. The common values for QP in H.264 encoder are between 18 and 28 and the default value is 23.

Let's state some other encoder's parameters used in the FFmpeg command during the conversations. The input stream is raw, i.e., without any header and padding information, but in YUV pixel format[3] with the frame rate and frame size (resolution) of 30 and 1280×720 respectively. The encoder reads its YUV stream from local memory after casting the stored mp4 video to YUV. Output rate was also set to 30 and *Baseline* was selected as its profile to avoid creating bidirectional (B) frames to obey the videoconferencing application requirements (Expert, 2021). Indeed, a B frame needs the next frame(s) to be encoded and decoded, which introduces increased delays, while the videoconferencing needs real time decoding and playback. Moreover, usually videoconferencing applications have less I-frames to decrease the bitrate consumption. So, the FFmpeg command in this experiment produces one intra (I) frame (as the first frame of each encoded version of a video chunk) followed by 29 predicted (P) frames. Finally, the output stream is in raw H.264 compressed format and ignores the overhead brought by RTP/UDP protocol headers for simplicity since such headers are small in comparison to the video data. The full FFmpeg command used for encoding is demonstrated in Appendix I.

2.2.3 Agent Unit

The agent unit is equipped with a decision method to determine an appropriate Quantization Parameter (QP) mode for the H.264 encoder such that it provides the highest possible QoE that the network conditions permit. Two algorithms are implemented: SARSA and BNN. For the main method, SARSA, the agent directly chooses a QP value. But, for the BNN model, it first selects a bitrate range then picks a relevant QP which falls in the range and is nearest to its target throughput (the center of the range). If there is no bitrate falling in the range, the QP with highest bitrate which is lower than the target throughput is selected. During the signaling phase, the agent selects a QP of 45 regardless of the method deployed.

[3] Although the sequences are identified throughout the Internet as in YUV format, they are actually in YCbCr format since YUV is used for analog video.

If the agent benefits from the SARSA method, it consumes the feedback coming from the receiver during the conversation in order to update the state-value function and explore or exploit among available actions. The BNN method is not involved in the agent-environment interaction to build its model, but uses existing data logged after a long-term experience. Afterward, the agent applies the BNN model to take action in the test phase of experiment.

2.2.4 Packetizer Unit

The output of the encoder is a raw encoded stream related to a time slot of the video. The packetizer unit fragments this time slot into packets corresponding to their frames, e.g., it generates 30 packets for a *ts* if the frame rate is 30, with the help of the FFprob command[4]. At the time of fragmentation, some other useful information for the receiver such as the frame's type (either I or P) and index of a packet among the stream in a *ts*, are added to each packet. It is assumed that each frame is sent using the UDP protocol which doesn't provide retransmissions.

2.2.5 Arrival Filter

At the receiver, arriving packets enter first into the arrival filter. This unit monitors some statistics of the packets like their delay, index of the first packet lost among a group of packets related to one time slot, delay gradients, and throughput. Moreover, packets received after the acceptable delay are omitted by this unit.

The first packet of the signaling phase sets the basis time at the arrival filter. The expected time for receiving the j^{th} packet in the i^{th} time slot during signaling or the conversation, $ExpT^{j}_{ts_i}$, is calculated with the help of both the basis time and time interval for each two subsequent packets that have been sent, i.e., $1/fps$. Eq. (2.7) describes this process. In Eq. (2.7), the time slot of the signaling phase is consumed as negative value just to emphasize it is separated from the main conversation which starts from time slot zero.

[4] https://ffmpeg.org/ffprobe.html

$$ExpT^{j}_{ts_i} = i + j \times (1/fps) \qquad i \in \{-8, -7, .., d-1\} \quad \text{and} \quad j \in \{0, ..., fps-1\} \tag{2.7}$$

where $d = 10$ is the video length (number of time slots).

The maximum time which the receiver waits for the j^{th} packet in the i^{th} time slot, $MExT^{j}_{ts_i}$, is no longer than tolerable buffer playout time ($PLoT$), which is a constant value of 40 ms, after $ExpT^{j}_{ts_i}$:

$$MExT^{j}_{ts_i} = ExpT^{j}_{ts_i} + PLoT \tag{2.8}$$

If a packet arrives later than $MExT^{j}_{ts_i}$, it is not involved in decoding. Whether this packet be an I frame or P frame, these late arriving packets are not playing a role in decoding, and consequently, the video on the user's screen freezes until the next decodable video chunk arrives.

Moreover, the arrival filter will check if a packet passes the Maximum Waiting Time ($MWT^{j}_{ts_i}$), which is $2\times$ $PLoT$ after the $ExpT^{fps-1}_{ts_i}$ as shown in Eq. (2.9). If a packet does not arrive by this time, it will be considered that the packet was lost in the network. The number of packets which do not play a role in decoding because they arrive late and the number of packets considered lost are integrated in the feedback signal.

$$MWT^{j}_{ts_i} = ExpT^{fps-1}_{ts_i} + 2 \times PLoT \tag{2.9}$$

The arrival filter also calculates the one-way delay gradient, q_{ts_i} for ts_i, as introduced in Carlucci *et al.* (2016). Delay gradient intends to capture the variety of the queuing delay by considering the difference between the time interval of two successive packets that arrived at the receiver and the time interval of their timestamps, i.e., their sending time, computed as:

$$q_{ts_i} = \frac{1}{N} \sum_{j=1}^{N} [(t_{j,i} - t_{j-1,i}) - (T_{j,i} - T_{j-1,i})] \tag{2.10}$$

where $t_{j,i}$ is the arrival time of j^{th} packet of the stream at time slot ts_i at the receiver and $T_{j,i}$ is the time that j^{th} packet has been sent as shown in Figure 2.10. N is the number of packets taking part in calculating the delay gradient of packets in the time slot ts_i. In the case of congestion in the network, q_{ts_i} would take positive value. The zero value of the delay gradient can be represented as no congestion in the network. Delay gradient is also one of the BNN's input to train its model.

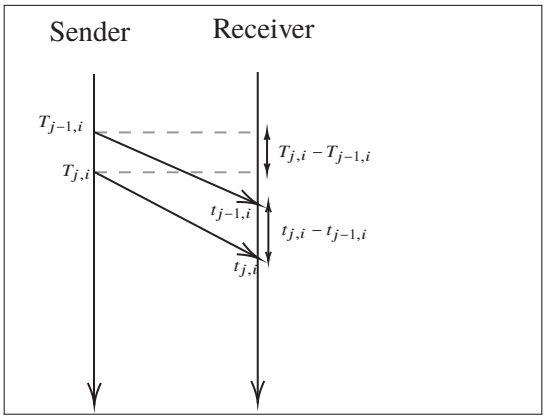


Figure 2.10 Delay gradient

2.2.6 Decoder, QoE evaluation, and Feedback

A packet that arrives after its maximum expected playout time is not being transferred to the decoder. As there is one I frame in a group of pictures at a time slot and the rest of the frames are P-frames, the decoder is unable to decode the rest of the frames after a lost video frame. It leads the video that has been played back on a user's screen to be frozen on the last frame received until the next time slot. The decoder produces the YUV format of the video with the same number of frames per second as the original video chunk by replicating frozen pictures. The output of the decoder is then compared with the YUV version of the reference video to calculate the PSNR values, i.e., PSNR for each frame in a *ts*.

The packet statistics obtained from the arrival filter (e.g., receiver-side throughput, loss rates, one-way delay gradient, etc.) and the Reward are encapsulated in a feedback signal to be transferred back to the sender. This signal can be carried through RTCP controlling message protocol (Schulzrinne, Casner, Frederick & Jacobson, 2003) in the real world.

The feedback then is used by the agent at the next time slot. During the experiment, it is assumed that the feedback is received by the sender no later than a timeout later. But in the real world, the agent can take its action according to the previous State observed if the feedback has faced delay or loss.

2.3 Network Simulation

As stated in the literature review chapter, the reason for applying automatic controller is to overcome the problem of conventional methods with rule-based architecture which are suffering from lack of generalization to the vast range of network types.

The Internet is good example of such diverse networks. While reviewing the literature in video transmission context, we found several works (such as Huang *et al.* (2018b); Mao *et al.* (2017)) deploying a trace-driven approach for their experiment for their training and test phases. They extract the traffic information of real users gathered in a small period of time, ranging from several hours to at most several days, to be used in their experiments. However, as the traffic information is gathered from a small period of time, this way is ineffective for our purpose of evaluating a method in a generalized environment. An experiment should be designed carefully to let one claim that a method is working in all network types (such as within Internet). In this project we could only theoretically design a way how to approach this generalization and we could implement a simple but a new classical-queuing-model-based network simulator. For solving the problem of integrating a method to diverse networks and evaluating its performance in that context, we divide it into sub-problems:

- Classify the network conditions of a general environment, like Internet, upon relevant and available characteristics of traffic flows like city, ISP, and server (as used in Sun, Yin, Jiang, Sekar, Lin, Wang, Liu & Sinopoli (2016)).
- If the number of classes is large, select a sufficient number of classes that represent the whole. If not, all classes are selected.
- Run experiments where each gets the agent to learn and evaluate over the network conditions that a selected class generates.

The agent in each experiment learns its own best model that can well work inside that class of network conditions. The performance of the method in these sample runs can be used to anticipate the performance of the method in the general network environment. We can aggregate what the agents learned toward shaping a main agent with multiple sub-agents which can be used when a corresponding class of network conditions are met (Zhang *et al.*, 2020). This is the general purpose that we see reasonable to follow. However, in this work, we could implement only a simple simulator. Also, it is difficult to set our network simulator to produce network condition of such class of network conditions. Investigating the behaviour of a sample class of network conditions, determining the sufficient number of classes which represent the entire classes (in the case of the number of classes be large), and finding appropriate simulator's configurations to produce each class (or finding other network simulators to do that) are valuable projects that should be followed up in future works.

The experiment, in this work, consists of several training and test trajectories. Each trajectory is considered as a separate virtual video call. In this process, for each trajectory, the sender picks a video from the database and encodes its contents at each time slot to send it over the established path on the simulated network to the receiver. Therefore, each trajectory establishes its connection over a distinct network path. Each network path behaves differently based on the traffic on the path and the wireless error rate. Specifically, in our experiments, the simulator's configuration is such that it produces network conditions belonging to two groups of users located in two fixed subnets as shown in Figure 2.12. Finally, we select two configurations

to produce low and high available bitrate network conditions as approximately two ends of conditions that these two groups of users within the subnets face. In the following, we discuss the topology and parameters deployed in each network's component. Also, we present the parameters' setting for one of our two configurations, which lead to network conditions with high available bitrate, that we applied during the experiments.

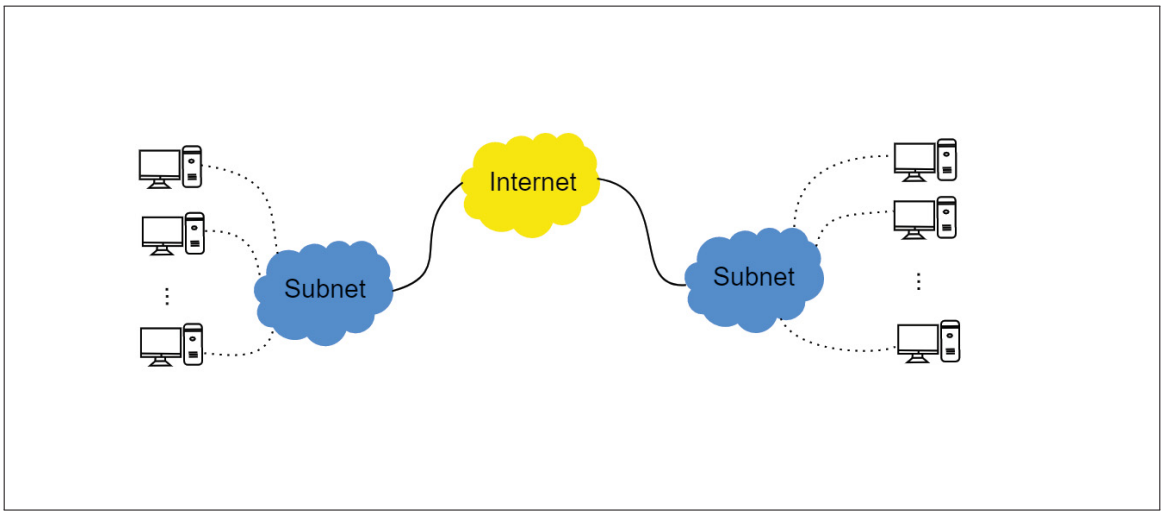


Figure 2.11 Two subnets connected to each other over the Internet. The experiment is performed with the assumption of conversations established between two end users located in two different subnets, which are fixed throughout the experiment

The simulator uses a network topology consisting of two wireless stations and two routers. The wireless stations are connected to the terminals, which host videoconferencing applications, at one end and to a router at the other end. A router is connected to both a wireless station and another router. Each network device is also connected to other nodes to receive other users' packets. The simulator gets the packet size and generates delays and packet losses.

A wireless station is composed of two units: bit-error checking and queuing pool. A packet subject to errors while transmitted over the wireless links will be detected by the first unit. The packet which faced no bit error, waits in the pool. The network is assumed to be a best-effort network. The available bitrate for our video flow for each trajectory strongly depends on other users' bitrate consumption. Hence, the available bitrate to a packet varies with time, which can

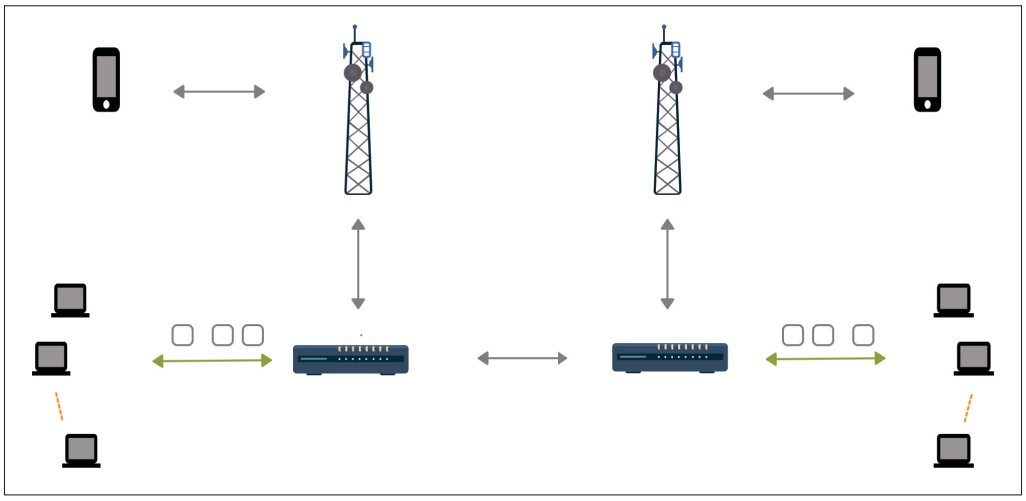


Figure 2.12 The network simulator's topology. The left wireless radio station establishes the wireless link 1 and the right one establishes a wireless link 2 to the terminals hosting the videoconferencing applications. The stations and routers are also assumed to be connected to other users

lead to variable delays for each packet. Delays can also occur when a packet is waiting in the queue of a router. The waiting time is affected by the other users in the network.

The wireless router has two main parameters: (i) bit error probability at link i, denoted as ρ_i, and (ii) the bitrate available for the video stream from our application at the k^{th} time slot, β_{ts_k}. The first parameter is randomly selected from a list indicating a small range of bit error probabilities ($\rho_i \in [1, 10] \times 10^{-9}$), then it is kept fixed throughout a trajectory. The latter parameter is randomly initialized from a list of bitrate values.

The wireless links are assumed to show a constant bit rate behavior (after interleaving and other transmission techniques) and a bit error probability of ρ_i for wireless link i. Hence, the probability of losing a packet of size $\mathcal{N}$ due to transmission errors for wireless link i, is:

$$P_i = 1 - (1 - \rho_i)^{\mathcal{N}} \tag{2.11}$$

If the bit error rate is small, it can be approximated by:

$$P_i \approx \rho_i \times \mathcal{N}. \tag{2.12}$$

The probability of receiving a packet of size $\mathcal{N}$ after passing over both wireless links is:

$$(1 - P_1)(1 - P_2) = (1 - \rho_1)^{\mathcal{N}}(1 - \rho_2)^{\mathcal{N}}, \tag{2.13}$$

and hence the probability of losing that packet would be:

$$P = 1 - [(1 - \rho_1)^{\mathcal{N}}(1 - \rho_2)^{\mathcal{N}}] \approx 1 - [(1 - \rho_1\mathcal{N})(1 - \rho_2\mathcal{N})] = \rho_1\mathcal{N} + \rho_2\mathcal{N} - \rho_1\rho_2\mathcal{N}^2. \tag{2.14}$$

For very small values of ρ_1 and ρ_2, and for $\rho_i\mathcal{N} \ll 1, \forall i$,we have:

$$P \approx (\rho_1 + \rho_2)\mathcal{N}. \tag{2.15}$$

To simulate the wireless links, a list of uniformly generated random numbers within the range of 0 and 1 with the length of the packet's size $\mathcal{N}_p$, for packet p, is constructed on each link. The packet is considered to be lost if one of its corresponding randomly generated numbers is below ρ_i. Otherwise, the packet is transferred properly to the next node (wireless router or terminal). Hence, the loss probability of our video packet highly depends on the size of the encoded video frame, also known as a transmission packet in this work. Moreover, the transmission packet size depends on other factors such as the frame resolution and the degree of dynamic scene of the frames.

The packet enters the bit error checking units, illustrated in Figure 2.13, after being transferred over a link. There, it will be discarded if any error occurs in the packet. Otherwise, it will enter the queuing pool. The wireless bitrate is shared among all users. Hence, the bitrate available to our application changes over time. This happens because our application does not have

any privilege compared to other users, i.e., no traffic management is provided and a simple First-In-First-Out (FIFO) is the assumed scheduling method used at the wireless router.

To simulate the bitrate, several parameters including available bitrate value at k^{th} time slot (β_{ts_k}), the time to stay on an available bitrate value (Θ_β), the step size used to change the bitrate (Δ), and the time duration to keep the selected step duration (Θ_Δ), are used. Available bitrate is randomly initialized in (8 Mbps, 10 Mbps) range, and then updated smoothly after a while (which is determined by Θ_β), with an adjustable parameter Δ. This Δ is also changed after it stays on a value over a time span (Θ_Δ).

Each parameter is selected randomly from a uniform distribution in its corresponding range as follows:

$$\beta_{ts_i} \in [8, 10] \text{ Mbps}$$

$$\Theta_\beta \in \{1, 2, 3\} \text{ second(s)}$$

$$\Delta \in \{100, 120, 140, 160, 180, 200\} \text{ kbps}$$

$$\Theta_\Delta \in [1, 5] \text{ second(s)}$$

It should be mentioned that these values have been set for simulating the transmission of typical videoconferencing of the High Definition (HD or 720p) resolution, i.e., 1280×720, and a certain level of scene dynamics. In other words, it is important for some parts in the network simulator (e.g., the wireless stations mentioned above) to rely on certain video content properties, like frame resolution and the dynamic characteristics of the video, to work as expected. For example, if the size of the video frame was in Common Intermediate Format (CIF) resolution, i.e., 352×288 or if the video was categorized as high degree in dynamic scene, the configuration of wireless network's bitrate (the range of values assigned) has to be changed before starting the experiment to produce certain network conditions (e.g., delay and loss rate). If the network conditions are too favorable, there won't be any transmission issue and there is no problem to solve. If they are too unfavorable, the application cannot operate at all. There is a specific range of conditions, although they are wide, where it is challenging to make the videoconferencing

application work while feasible. In this work, we determine two network conditions of low and high available bitrate by careful manual investigation of potential ranges for the current settings. The parameters' values brought in the Network Simulation section represent one of our two configurations (see section 3.1.2) which are considered to produce network conditions with high available bitrate applied during the experiments.

It is also worth mentioning that the minimum operating time cycle in the simulator is $1/fps$, i.e., the time for each video frame, also known as packet in this work, to be transferred in the network. Whenever a packet enters a node, it triggers the execution of that node to calculate its network statistics, like delay, based on the node's parameters, e.g., bitrate. Once a parameter is set, it will be used at least in one time slot, while each time slot executes *fps* number of video packets.

Consider the $(j-1)^{th}$ packet, where $j > 1$, of time slot i, $P_{ts_i}^{j-1}$, arrives at the wireless router's queue, it waits for a certain time to get served completely according to the bitrate availability. Meanwhile, it is possible for the next packet, $P_{ts_i}^{j}$, to arrive in the queue while the current packet is being served. So, the time interval between these two packets is required to calculate the delay, $Delay_{ts_i}^{j}$, of the coming packet, $P_{ts_i}^{j}$. Delay is calculated as:

$$Delay_{ts_i}^{j} = \mathcal{N}_{ts_i}^{j} / \beta_{ts_i} + Overhead \tag{2.16}$$

$$Overhead = \begin{cases} Delay_{ts_i}^{j-1} - Interval^{j} & , \text{ if } Delay_{ts_i}^{j-1} > Interval^{j} \\ 0 & , \text{otherwise} \end{cases}$$

where $\mathcal{N}_{ts_i}^{j}$ denotes the length of the j^{th} packet (in bits) for the time slot i. $Interval^{j}$ is the time interval between the arrival of the j^{th} packet, $P_{ts_i}^{j}$, and the previous packet, $P_{ts_i}^{j-1}$, at that node. If a packet is the first packet in the encoded stream of ts_i, then $P_{ts_i}^{j-1}$ and $Delay_{ts_i}^{j-1}$ point to the last packet of the previous time slot, ts_{i-1}. *Overhead* is considered zero for the first packet of the first time slot in a trajectory.

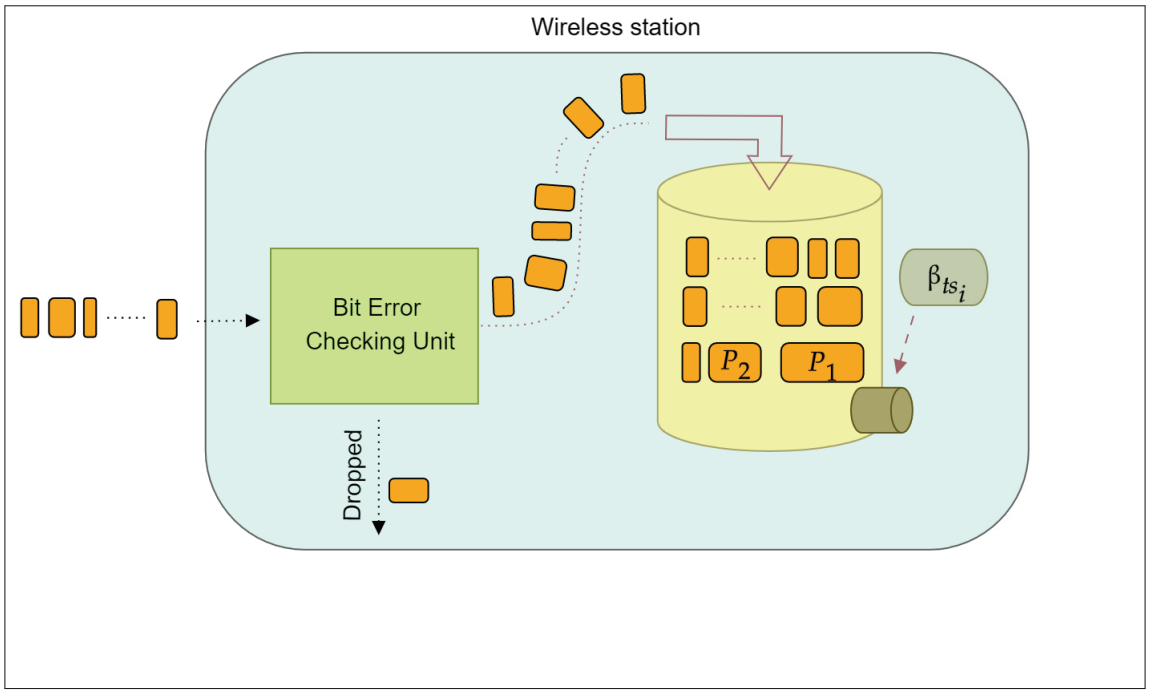


Figure 2.13 Video packets enter the wireless station (or router). First they are monitored in the bit error checker unit to discard the incorrect packets. Surviving packets are transferred to the pool where the packets wait to be served. The bitrate at the current time slot and the time interval between the current packet entering the pool and the previous packet are the main two factors playing a role in modeling the delay encountered by the current packet

The wireless stations are connected to two routers each of which are assumed to have infinite queuing capacity, i.e., there is no discarding policy regarding the maximum buffer size. Each router is assumed to be connected to several other nodes (terminals, wireless stations, routers, etc.), as it is shown in Figure 2.12. Therefore, the video application's packet is received by the routers along with other packets. Hence, these packets are the information units serving the routers.

Routers are scheduled based on simple FIFO without priority. At each time slot a bunch of packets enter the router's queue and are stacked on top of other packets which arrived at the previous time slots and waiting to be served as shown in Figure 2.14.

The classical queuing model is used to evaluate the delay of each video packet passing through the router. Therefore, all the packets arriving from other resources in each time slot, χ_{ts_i} (i.e., before the arrival of each video packet) along with the serving rate, μ, are important in the process of calculating the delay. χ_{ts_i} is randomly selected around a predefined average value, λ_{ts_i} by applying the Poisson law[5]. Hence, the modeled delay is a stochastic process which mainly depends on the serving rate and the average number of packets arriving from other resources at the router in a time slot, λ_{ts_i}.

The average number of packets arriving at each time slot is kept for θ_λ seconds before it gradually changes (increases or decreases) by δ to produce a dynamic aspect of the delay. The range of these parameters is set as follows:

$$\begin{aligned} &\mu = 1000 \text{ packets per second} \\ &\lambda_{ts_i} \in [1/4\ \mu, 3/4\ \mu] \\ &\theta_\lambda \in [1, 7] \text{ second(s)} \\ &\delta \in \{10, 30, 50, 70, 90\} \text{ packets per second} \end{aligned} \tag{2.17}$$

In the changing process, a random value is selected from a uniform distribution in the $[1/4\ \mu, 3/4\ \mu]$ range. If that value is more than the current average λ_{ts_i}, the average will be increased by δ but cannot exceed the upper-bound, $3/4\mu$. Otherwise, the average value will be decreased, but cannot go below the lower-bound, $1/4\mu$. The changing value, δ is maintained for θ_δ seconds, which can be randomly picked from the following range:

$$\theta_\delta \in [2, 10] \text{ seconds} \tag{2.18}$$

[5] See the distribution in NumPy library that we used https://numpy.org/doc/stable/reference/random/generated/numpy.random.poisson.html

It is important to note that the total number of packets coming from other resources, χ_{ts_i}, is obtained by applying Poisson law around λ_{ts_i}. To calculate the delay encountered by each video packet, the simulator needs to have the details about the number of packets coming from other sources before the time of each video packet arrival. The number of packets in front of each video packet is determined by applying Poisson law around (χ_{ts_i} / fps) at the time that each video packet enters the queue.

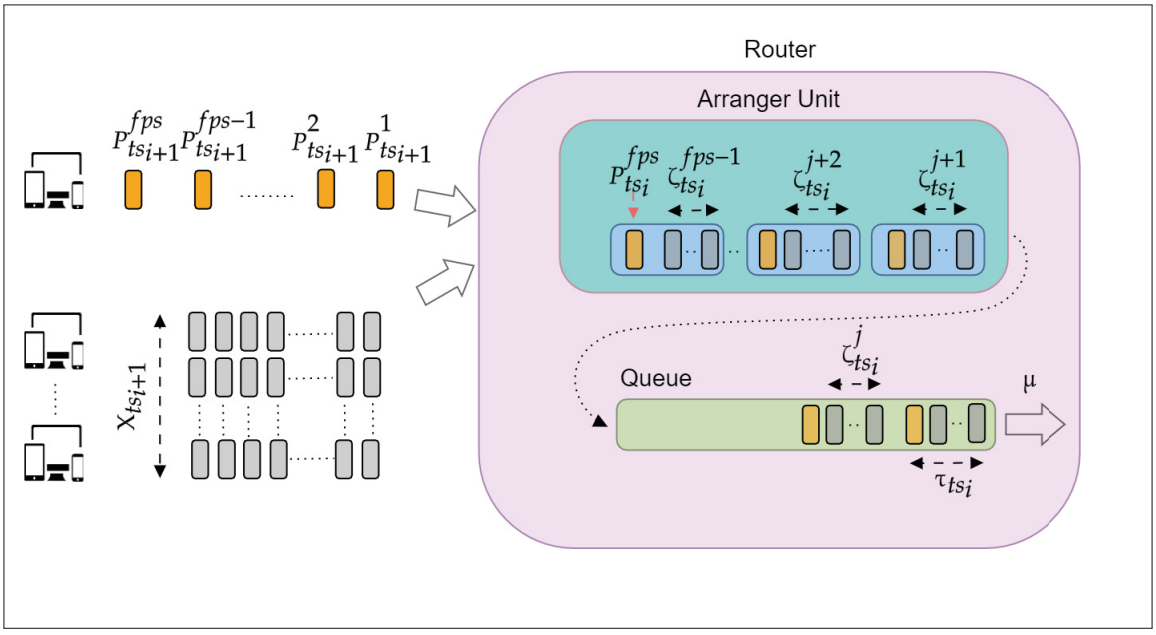


Figure 2.14 The process of modeling the delay that each video packet encounters. The arranger unit, randomly distributes χ_{ts_i} packets coming from other resources among our video packets entering the router. $\zeta^{j}_{ts_i}$ denotes the number of packets coming from other resources before the j^{th} video packet and is calculated by applying Poisson law around the (χ_{ts_i} / fps) as the average value. Both the number of packets already present in the queue and the number of packets newly arrived before a video packet entering the router take part in modeling the delay encountered by the new coming video packets

Again, it is worth mentioning that, we also take the time interval between two subsequent video packets transferred over the network into account to calculate the modeled delay of each video packet. The length of this time interval depends on how fast the previous node (terminal or a router) has been able to send (or consume) a packet in the current node. The time interval

is important because it lets the model consider the possibility of having remaining packets from the previous time cycle in the queue. In this regard, the simulator does not discard a packet containing bit-errors but delivers them (virtually) to help the simulator calculate the time interval.

The network simulator, therefore, obtains video packets and passes them to the receiver. On the way, a packet may get lost or face delay. A packet might be lost (virtually) only due to the bit error at the wireless station. The routers do not create any packet loss. The delay each packet faces depends on the available bitrate of each wireless station, the size of the video packet which is intended to be delivered, the number of packets coming from other resources at each router, and the number of packets remaining in the queue from previous time cycles.

The simulator is used in two types of experiments. First, it is deployed in an experiment which implements the proposed SARSA method. In this case, each trajectory, e.g., the training or the test one, launches a new simulator which logs traces to be stored. Launching a new simulator means that all main parameters mentioned above are randomly initialized before the establishment of a trajectory. Second, the simulator will be part of the experiment run for the baseline BNN method. In this case, the simulator consumes the stored traces and repeats the parameters using the same strategy as the main experiment to provide the same network conditions, but behaves differently based on the baseline method's choice at each time slot. Hence, it might yield a different network performance based on the baseline's choice. The reason behind implementing this approach is that it provides us with a fair performance evaluation for each compared method.

2.4 Statistical Analysis

In this research, parametric tests will be used to evaluate the significance of the observed effects (the effects of observed data gathered as the results of the experiments). Indeed, the significance of our performance results will be analyzed using such parametric tests in section 3.2.3. Particularly, we adopt Analysis of Variance (ANOVA) as the main statistical test to evaluate the effect of

main factors and their interactions in the results. ANOVA (Fisher, 1992) is a tool to analyze the variances among three or more means by comparing the variances (σ^2) both within and across groups.

Our data has two factors: i) network type (factor A), and ii) method (factor B). Each has two treatments: network condition with low and high available bitrate as network types ($b = 2$ treatments) and SARSA and BNN for methods ($a = 2$ treatments). For each treatment, there are $n = 10000$ stored observations (except for BNN over network condition with high available bitrate which is 9450 due to some failures), y_{ijk} where $i = 1, 2, .., a$, $j = 1, 2, .., b$, and $k = 1, 2, .., n$, at the resolution of time slot from the test trajectories. Each observation can be written as

$$y_{ijk} = \mu_{ij} + \epsilon_{ijk} \text{ with } \begin{cases} i = 1, 2, \ ..., \ a \\ j = 1, 2, \ ..., \ b \\ k = 1, 2, \ ..., \ n \end{cases} \tag{2.19}$$

$$\mu_{ij} = \mu + \tau_i + \beta_j + (\tau\beta)_{ij}$$

where μ is the overall mean, τ_i is the effect of the i^{th} treatment of factor A (say network types), β_j is the effect of j^{th} treatment of factor B (say method), $(\tau\beta)_{ij}$ is the effect of the interaction between τ_i and β_j, and ϵ_{ijk} is an error component. Error component tries to incorporate all other sources of variability in the experiment. In two-factor factorial, both factors are of equal interest and the test hypothesis is determined as equality of each treatment effect in each factor and their interactions:

$$H_0 : \tau_1 = \tau_2 = ... = \tau_a = 0$$
$$H_1 : \text{at least one } \tau_i \neq 0 \tag{2.20}$$

$$H_0 : \beta_1 = \beta_2 = ... = \beta_b = 0$$
$$H_1 : \text{at least one } \beta_j \neq 0 \tag{2.21}$$

$$H_0 : (\tau\beta)_{ij} = 0 \text{ for all } i, j$$
$$H_1 : \text{at least one } (\tau\beta)_{ij} \neq 0. \tag{2.22}$$

Given the observations, ANOVA tests these hypotheses by calculating F value, which is the significance of main effects and interaction, or *P-value*. ANOVA finds the F value by dividing the corresponding *mean square* by the *error mean square*. Large value of this ratio shows that the observations do not support the null hypothesis. In this work, we assume the F value of 100 or above as large value. ANOVA has two other parameters which contribute to calculating mean square value. These parameters are *sum of squares*, *degree of freedom* which are obtained for all factors and their interactions. These statistical descriptions are summarized in an analysis of variance shown in Table 2.1. For details in formulation, the reader is referred to section 5.3 of Montgomery (2017).

Table 2.1 The Analysis of Variance Table for the two-Factor Factorial (brought from Montgomery (2017))

Source of Variation	Sum of squares	Degree of Freedom (df)	Mean Square	F
A treatments	SS_A	$a-1$	$MS_A = \frac{SS_A}{a-1}$	$F = \frac{MS_A}{MS_E}$
B treatments	SS_B	$b-1$	$MS_B = \frac{SS_B}{b-1}$	$F = \frac{MS_B}{MS_E}$
Interaction	SS_AB	$(a-1)(b-1)$	$MS_{AB} = \frac{SS_{AB}}{(a-1)(b-1)}$	$F = \frac{MS_{AB}}{MS_E}$
Error (or Residual)	SS_E	$ab(n-1)$	$MS_E = \frac{SS_E}{ab(n-1)}$	
Total	SS_T	$abn-1$		

The *P-value* in the table also represents, in addition to F, whether the null hypothesis for a factor is met or not. The smaller the *P-value* the stronger one can reject the null hypothesis. We consider, here, a *P-value* of less than 0.01 as a significant value to reject the null hypothesis for the factor. So, it means that there is a treatment in the factor such that its effect is not zero. Therefore, the distribution of the observations for each treatment of a factor is not the same and it suggests that the researcher should go further in analysis by comparing various pairs of means using a Post hoc comparison of means (i.e., Post hoc Turkey's honest significant difference test in this project as suggested by Jamovi software [6]). Post hoc Tukey's honest significant difference (HSD) test is used to correct for multiple comparisons done between groups of observations (Tukey, 1949). Post hoc Turkey's test has *Ptukey* column which represents whether the null hypothesis is supported or not. We consider the *Ptukey* value of less than 0.01 as a significant value to reject the null hypothesis for the interaction.

In the performance assessment section, we use ANOVA to check the extent of which the distributions of observations belonging to treatments or interaction are really distinguished. In our analysis, we also perform further analysis only for the interaction of the factors to compare their mean differences and the similarity of the distribution of each set of observations belonging to different combinations of treatments.

2.5 Summary

In this work, we formulate the source-rate control problem in RL in a new perspective and in a format precisely designed for videoconferencing systems. We deploy a tabular one-step TD reinforcement learning method, i.e., on-policy SARSA, as a source rate controller to take control of QP parameter of an H.264 video encoder. To the best of our knowledge, we represent the first tabular RL method for real-time interactive video transmission, in particular for videoconferencing. The problem is formulated by determining the agent and its environment along with defining state, action, and reward signals to let the agent interact with its environment and learn its optimum policy. The state signal consists of four parameters, where three of them

[6] Jamovi is a statistical analysis software which is available here https://www.jamovi.org/download.html

relate to network-related status and one is the sending bitrate. The action signal selects the QP parameter of the encoder and gets values among several QPs. The reward signal is obtained by a function of PSNR, a full-reference visual quality metric, as a representation of viewer-side perceived video quality. Evaluation of the proposed method in generalized environment was not available. Thus, after suggesting a way for anticipating the method's performance over a wider range of network types, we present a new classical-queuing-model-based network simulator as an initial approach. In this work, we have two configurations of the network simulator to produce low and high available bitrate with the aim of representing two-end conditions that two groups of users under the fixed subnets face.

CHAPTER 3

EXPERIMENTS AND RESULTS

In this chapter, we present the results of our experiments to validate the performance of the proposed SARSA-based method for rate control. We first introduce the videoconferencing database and identify the configurations of the network simulator which we are using for the experiments. Then, we provide the results of the evaluation performed after a training phase for both our SARSA-based and BNN-based[1] source-rate control over two low and high available bitrate network conditions. In this chapter we use the keywords of *"Simple"* for high available bitrate and *"complex"* for low available bitrate.

3.1 Dataset

The dataset for this work consists of the network related data generated over the experiment and the video database used to be transmitted over network simulator. Specifically, we use a video database containing visual content that is then encoded (compressed) and packetized at the sender, transmitted over the network simulator inducing delays and losses, and then received, decoded and rendered for QoE evaluation at the receiver as illustrated in Fig. 2.8. More details on the video files and the simulations's configuration are provided in the following sub-sections.

3.1.1 Video Files

The video files for the simulations are selected from YouTube datasets, i.e., YouTube-8M [2]. In the selection process, all videos with the frame size of 720p and videoconferencing contents, which met the following criteria, were extracted:

- Similarity in the number of people in front of the camera (i.e., one person in this work);

[1] A recent deep learning-based method used for this field (Huang *et al.*, 2018a).

[2] A Large-Scale Video Classification Benchmark at https://research.google.com/youtube8m/index.html after the work of Abu-El-Haija, Kothari, Lee, Natsev, Toderici, Varadarajan & Vijayanarasimhan (2016).

- Similarity in users' conducts (e.g., regarding the user's body activity, which yields scene dynamics. For example one set of video clips was taken from an instructor standing in front of the whiteboard and often move his head back and forth between whiteboard and camera. Another set of videos was taken from one person sitting on a chair and most of his activities were limited to his mouth's movements). In this work, we selected videos with a person sitting on a chair with possibly similar behaviors;

- Similar state of the camera (e.g., fixed state regarding the rotation or any camera's movement). We used videos without camera rotation or any movement;

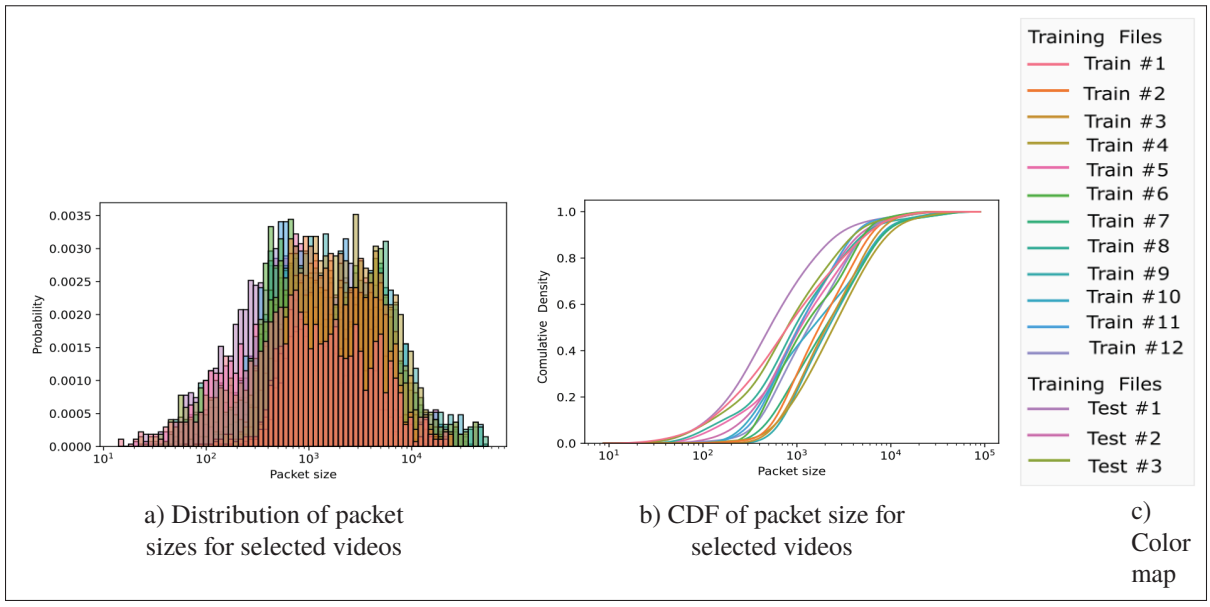
- Similar background (i.e., constant background in this work).

These criteria were defined to provide similar baseline in terms of video characteristics for training and testing. We selected similar baseline for two reasons. First, the data which the application is dealing with is not limited to the video file but mostly focuses on the network simulator's conditions. Selecting the heterogeneous input video can impact the current data space and is not representative of the desired videoconferencing application. Second, the application, in this project, is unable to sense the video characteristics and using similar input video, regarding the nature and characteristics, can lead the agent concentrate more on the changes that happened in the network. Nevertheless, the videos selected were representative of videoconferencing content. After reviewing the potential video clips from the YouTube-8M website, a total of 15 videos met these conditions, twelve of which were randomly selected for producing thousands of simulations for training, and three remaining files were used in thousands of test trajectories production.

To speed up the experiments, different versions of encoded streams, according to their Quantization Parameters (QP)[3], from the video files were stored. Therefore, throughout the experiments, compressed streams are read and transferred over the network. Each encoded stream consists of video frames covering the period of 1 second. There are fps (i.e., 30) such frames in each stream. The first packet of a stream is an Intra (I) frame and its size is much bigger in terms of

[3] See https://trac.ffmpeg.org/wiki/Encode/H.264

data size than the rest of the packets, which are predicted (P) frames. Considering all encoded streams, the distributions of the packets' size belonging to a video file are depicted in Figure 3.1.



a) Distribution of packet sizes for selected videos

b) CDF of packet size for selected videos

c) Color map

Figure 3.1 Packet size (KB) distribution of video clips encoded by different CRF (i.e., 20, 25, 30, 34, 40, 45) used. a) count of packet sizes, b) cumulative distribution of videos' packet sizes

As Figure 3.1 shows, around 85% of the packets have a size of 10 KB or less and a little portion of packets have a size of 10 KB to 100 KB. Knowing such a distribution helps to set appropriate configurations for the network simulator with the goal of selecting the conditions which are relevant for such videos.

Figure 3.2 illustrates a snapshot, the packet sizes in each frame, and encoded video bitrate with different QPs for one sample video from train and test groups (first and second row, respectively). In the middle column, the pattern in P frame sizes represents the scene dynamic in a video chunk. Generally speaking, a video chunk (in one shot[4]) with more dynamic scenes have larger P frame size in its encoded stream, as we can compare two first rows with the third row in Figure 3.2. The first two rows in the figure represent the similarities in video dynamics level within our dataset.

[4] See https://en.wikipedia.org/wiki/Shot_(filmmaking)

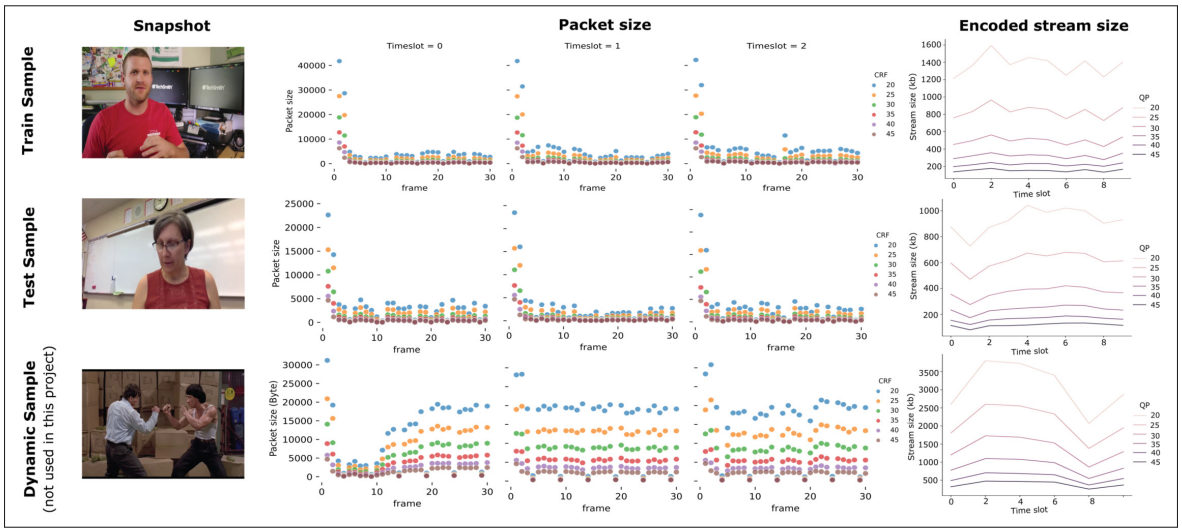


Figure 3.2 Snapshot, packet size and video stream size of three video samples. The first two rows belong to two video samples of our video database. The third row is a video containing dynamic scenes (not used in this study). First column: snapshot, second column: packet size (KB) of corresponding video clip encoded by different QPs (i.e., 20, 25, 30, 34, 40, 45 - illustrated with different colors) at first three time slot (horizontal axis in each graph). Third column: video stream size after encoding with different QPs for each time slot

3.1.2 Network Simulator's Configurations

As discussed in detail in the section 2.3, there are several parameters that can be set to define the network conditions. In this work, we evaluated our method in two scenarios: a simple network situation, where we have high available bitrate which leads to lower delay and losses; and a complex network with higher levels of delay and loss. The parameters used for these two scenarios are brought in Table 3.1.

3.2 Experimental Evaluation

Our proposed model is trained with the details explained in section 2.1.4. In the training phase, the proposed model, SARSA, is trained within the conversations, and the baseline, BNN, is trained offline using the data extracted from the terminated conversations, say, trajectories. The

Table 3.1 Network parameters in simple and complex network conditions

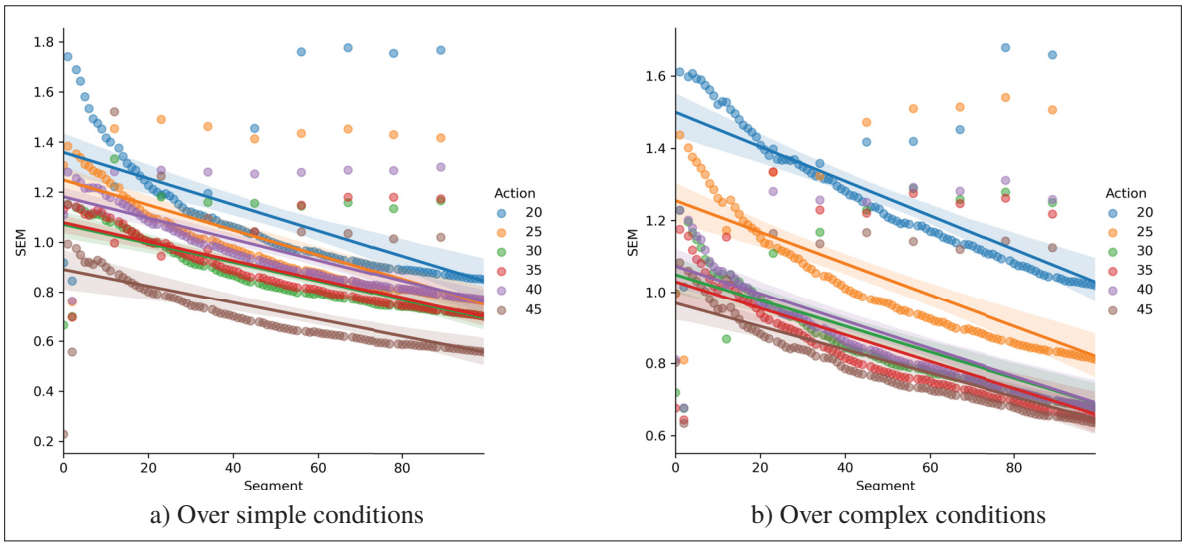
Network parameters	Value(s) or Range
Wireless	
ρ_i	$[10^{-9}, 10^{-8}]$
β_{ts_i}	[8 , 10] Mbps, *As Simple condition* [4 , 6] Mbps, *As Complex condition*
Θ_β	{1, 2, 3} second(s)
Δ	{100, 120, 140, 160, 180, 200} kbps
Θ_Δ	[1 , 5] second(s)
Routers	
μ	1000 packets per second
λ_{ts_i}	[1/4 μ , 3/4 μ]
θ_λ	[1 , 7] seconds
δ	{10, 30, 50, 70, 90} packets per second
θ_δ	[2 , 10] seconds

following sections bring the results gathered from training and test phases of the experiments. Note that the Seaborn Python library was used to generate the figures. As mentioned in sections 2.1.4 and 2.1.5, the models are trained using 20,000 trajectories and tested using 1000 new trajectories. Different videos are used for training and testing.

3.2.1 Convergence of our Proposed RL Method during Training

To evaluate proper training of our model, we checked its convergence over the course of training. The approach used for this process is based on calculating the standard error of mean (S.E.M.) for each state action combination until the end of each training segment. Samples in a segment, here, is the value of each state action combination in state-action value function ($Q(S_t, A_t)$) after each update occurred from the beginning of the experiment until the end of the segment. The S.E.M. estimates how far the mean of samples is from its true population mean (Andrade, 2020). In the case of proper convergence, it is expected that the S.E.M. decreases as we progress through training epochs (i.e., segments here). Figure 3.3 demonstrates the convergence pattern of two sample states in state-action value function each visited >10K for a simple and complex

network. Both figures for all its actions show that the distance of the samples means in a segment is approaching to its true population mean as the training phase progresses over segments.



a) Over simple conditions b) Over complex conditions

Figure 3.3 Training convergence for two sample states in state-action value function in run over a) Simple and b) Complex network conditions. The actions represent different QPs. Given an state action pair, each point represents S.E.M. of updates that the pair encounters from the beginning of the experiment until the end of its corresponding segment. The true population means are shown in solid lines

It is important to bear in mind that in Figure 3.3, there are very few outlier samples during training which result from the randomness (from video data and the network) in the training process.

3.2.2 A Sample in Test Trajectories

To perform a detailed inspection of functionality of the system (especially the network simulator), we randomly select a sample test trajectory, the trajectory $id = 77$ out of 1000, for two methods over two network conditions. In each figure presented in this sub-section, there are six sub-figures: i) *delay* for each packet (frame) encountered in seconds; ii) the overall number of packets that entered in both routers and that stand in front of our multimedia packets; iii) the size of each packet along with the stream size and available bitrate; iv) the number of packet losses due to

exceeding a tolerable time to be decoded (i.e., $MExT^{j}_{ts_i}$ mentioned in 2.2.4) at each time slot; v) the number of packet losses due to exceeding a maximum waiting time for packing the feedback; and vi) the PSNR of each packet after decoding at the receiver. Figures 3.4 and 3.5 are related to SARSA and BNN over simple condition. Figures 3.6 and 3.7 are corresponding to SARSA and BNN over complex condition, respectively.

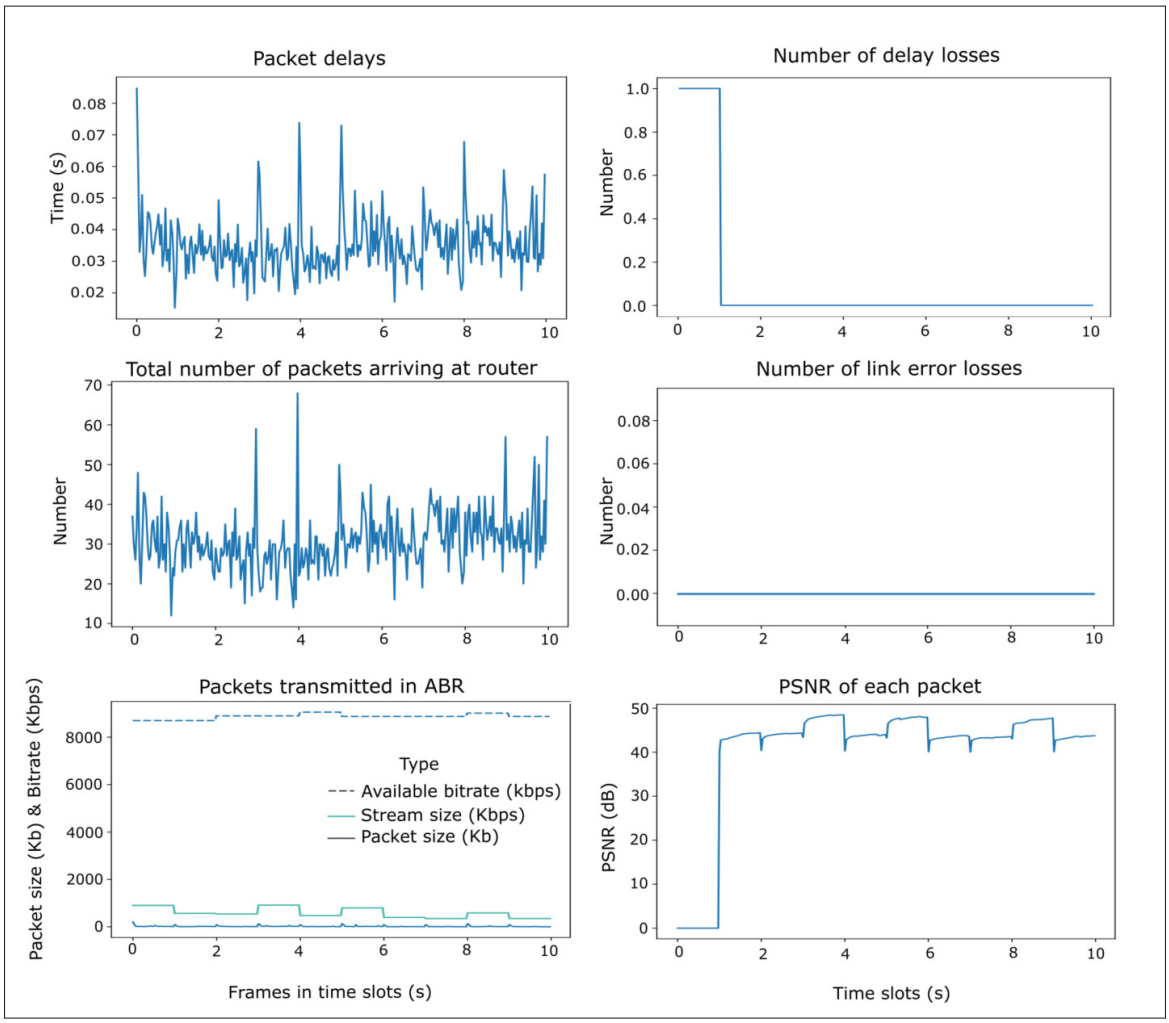


Figure 3.4 Details of a sample trajectory from SARSA over simple condition

During a trajectory, the delay that a packet faces depends on five factors: i) the size of the packet, ii) available bitrate, iii) the number of packets in front of it at the routers, iv) the time spent for the previous packet to be served by the routers, v) and the packets arrival-time interval. The

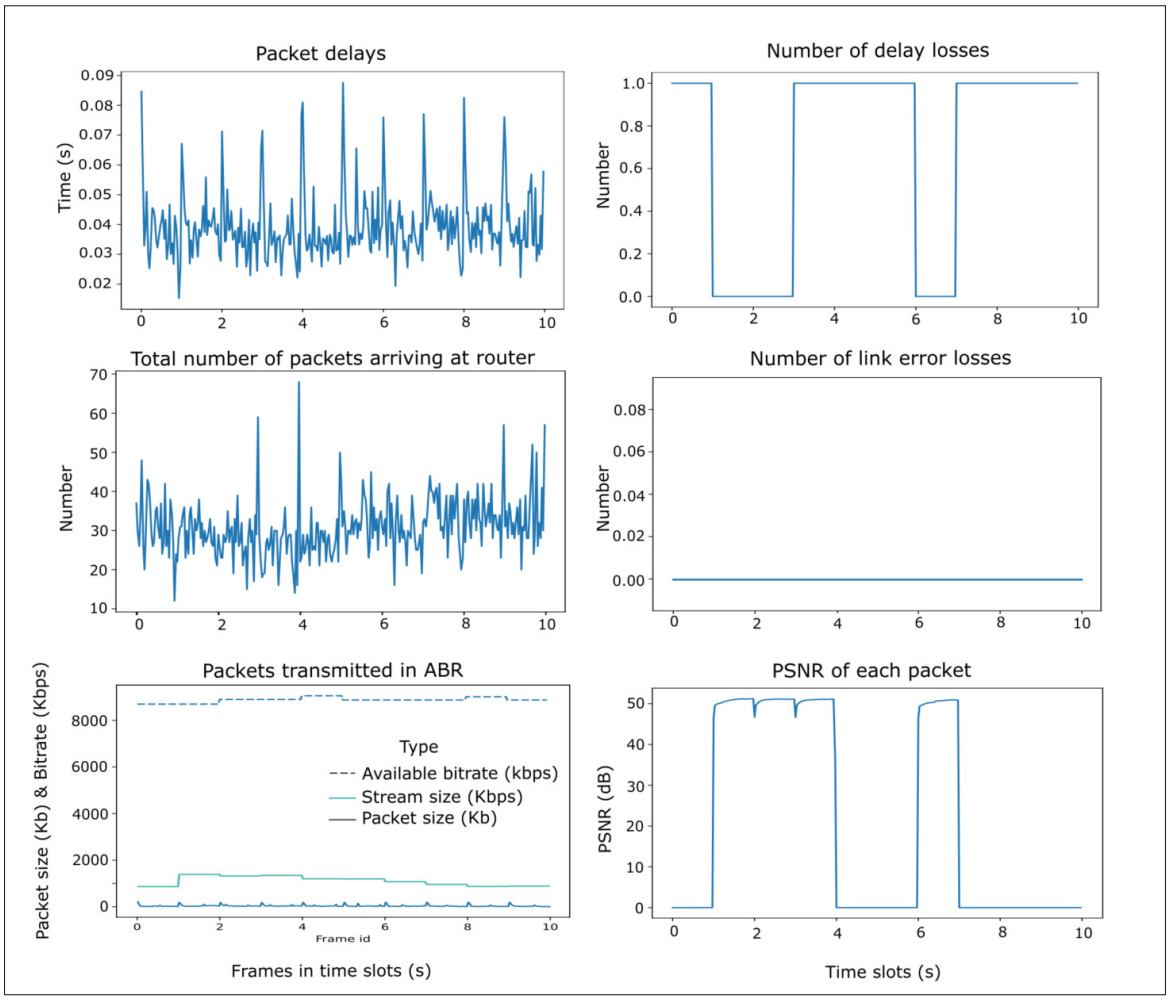


Figure 3.5 Details of a sample trajectory from BNN over simple condition

figures show well these dependencies. However, the effect of packet size and number of packets arrived in front of our packet at the routers are more significant.

Sub-figures related to packet losses due to the delay are just affected by the delay that a packet faces. The number of packets at each time slot that are exceeding their tolerable time to be decoded is depicted. Sub-figures related to packet losses due to exceeding the maximum waiting time are affected by both the packets which have long delays and the packets that had bit flipped (i.e., due to link error). Finally, PSNR sub-figures are affected by both the size of the stream after encoding and the index of the first packet that has been lost. If the first packet of a time slot is lost, no packet can be decoded at the decoder, even if all $fps - 1 = 29$ following packets

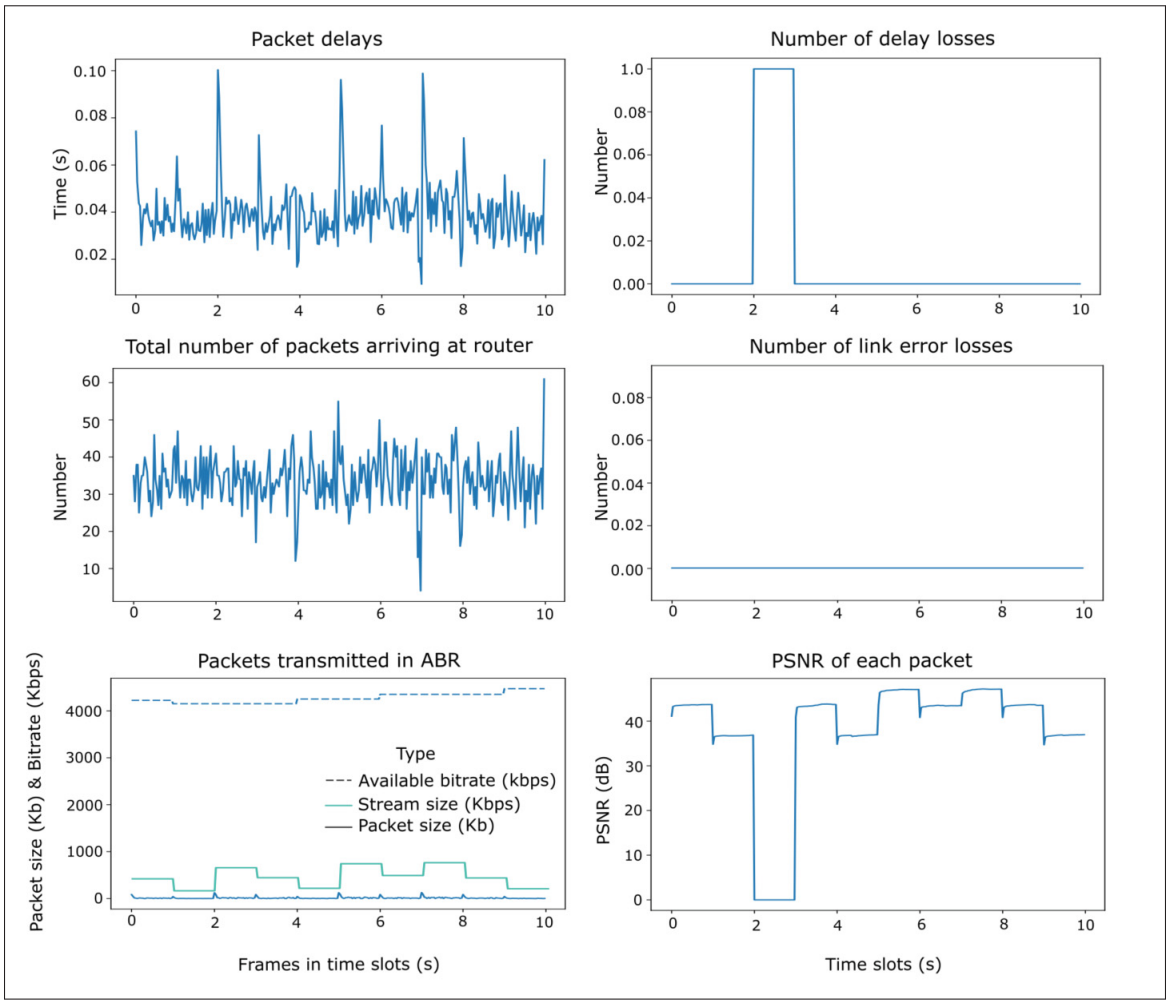


Figure 3.6 Details of a sample trajectory from SARSA over complex condition

arrived on time. In this case, the PSNR of all frames belonging to that time slot will be zero. If one middle packet of a time slot is lost, i.e., among all $fps - 1 = 29$ packet, then the last packet received is played back until the end of the time slot. Because the middle packet in that time slot is only depending on its previous packet to be able to be decoded. So, generally speaking, the extend of which the first index of lost packets is close to the last packet of the time slot, the PSNR of played back stream is approaching to the PSNR of the decoded version of encoded stream at the sender. The figures above show well these effects in a sample trajectory id for SARSA and BNN method over simple and complex scenarios. As the network conditions is

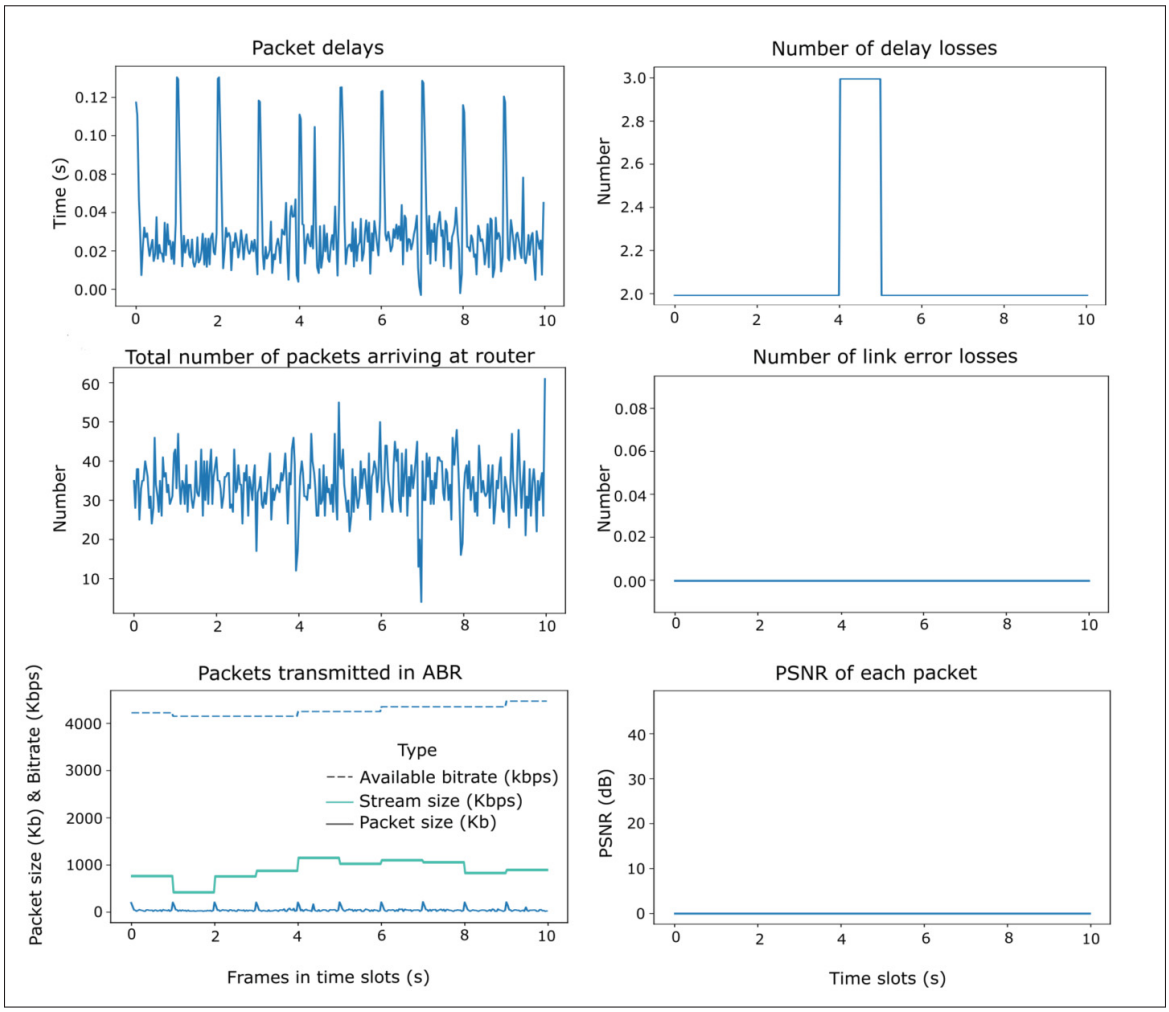


Figure 3.7 Details of a sample trajectory from BNN over complex condition

mainly repeated for both SARSA and BNN, it can be seen that the same total number of packets arrived at the router along with the same bitrate behavior.

While the packet size is much smaller than the available bitrate in wireless station, we can observe that the effect of the packet size in adding the loss and breaking the quality (PSNR of zero) is significant. One reason is that the tolerable time to receive a packet in videoconferencing system is limited. Another reason is due to the much larger size of intra frames compared to inter frames. This increases the chances of intra frames to be lost and consequently to compromise the quality. In order for the intra frames to arrive on time for display, the stream bitrate needs to

be reduced. A solution to use more effectively the bandwidth would be to select a higher QP for intra frames (and reduce their size) or allow a larger jitter buffer, but this latter solution is not practical for real-time videoconferencing. We can also observe that both method perform better over simple condition than complex condition and that SARSA performs much better than BNN in conditions but significantly better for the complex case.

3.2.3 Performance Assessment

In the next step, we compare the performance of our approach with the BNN baseline in both simple and complex network scenarios. Three measures are used for the assessment: peak signal-to-noise ratio (PSNR), packet loss rate due to delays, and packet loss rate because of link errors.

To reduce the bias, a set of fixed testbeds is designed for the baseline and our method by fetching the same video input and launching the same network behavior. Our test contains 1000 trajectories for both SARSA and BNN methods which are performed after training the models. As previously explained, the experiment were ran over two network configurations to compare the methods in different testbeds. Details of each assessment are explained in the following.

3.2.3.1 PSNR

PSNR represents the peak signal-to-noise ratio as defined in Eq. (2.3). During the test phase of 1000 trajectories, we gather the average PSNR over the frames of each time slot for each trajectory. Figure 3.8 shows this value for both scenarios with respect to the network complexity (see section 3.1.2) which shows that our SARSA-based technique outperforms significantly BNN. For the simple network scenario, the average PSNR for SARSA is 42.2 dB while it is of 35.5 dB for BNN; a difference of 6.7 dB. For the complex network scenario, SARSA provides a slightly smaller PSNR than for the simple case at 38.8 dB while the BNN simply fails to deliver the video streams as shown in Table 3.2. To show the significance of these results, we perform the two-factor Analysis of Variance (ANOVA) test (as briefly introduced in 2.4) which is used to

evaluate potential differences between the distributions of PSNRs (gathered from all time slot of all test trajectories) in two methods and two network simulator configurations (see section 2.4).

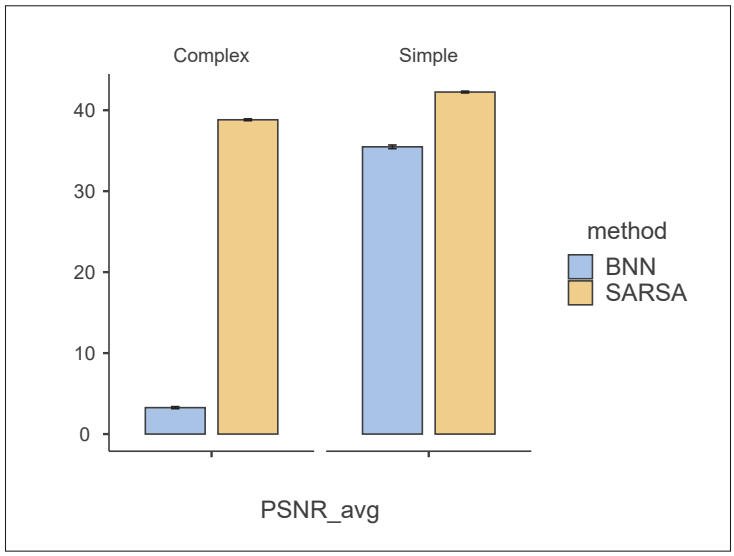


Figure 3.8 The average PSNR results, in dB, for SARSA (yellow) and BNN (blue) methods gathered from 1000 test trajectories and over both simple and complex networks after 20k training trajectories

Table 3.2 Mean and Standard deviation of PSNR averaged over frames in each time slot for the methods over two network conditions

Method	Network Condition	PSNR (dB)
BNN	Simple	35.5 ± 0.2
	Complex	3.27 ± 0.12
SARSA	Simple	42.2 ± 0.1
	Complex	38.8 ± 0.1

A two factors (Method x Network Complexity) between subject ANOVA conducted on the data yielded a significant interaction, $F(1, 39446) = 8959$, $P < 0.001$. It means that there are at least two sets of observations (among pair of treatments from method and network type factors) that do not have the same distribution. See Table 3.3 for statistical details.

Table 3.3 ANOVA on quality of experience (PSNR). Main effect of method and network conditions, and their interaction's effect on observed PSNR. There are two classes for both factors (degree of freedom = 1), and a total 39446 observations. All effects are significant with P < 0.001

	Sum of squares	df	Mean Square	F	P
method	4.41×10^6	1	4.41×10^6	19,362	< .001
NetType	3.13×10^6	1	3.13×10^6	13,731	< .001
method * NetType	2.04×10^6	1	2.04×10^6	8959	< .001
Residuals	8.99×10^6	39,446	228		

Post hoc Tukey's test was conducted to evaluate further the interaction between method and network complexity. As illustrated in Figure 3.8 and detailed in Table 3.4, all pairs of treatments do not have similar distributions. Mean Difference column of the table represents the difference of observations' mean between all two pairs of treatments (i.e., BNN, SARSA, simple, and complex). The table shows that the mean of observations for BNN over complex is less than: i) BNN over simple condition by 32.21 dB; ii and iii) SARSA complex and simple condition by 35.55 dB and 38.98 dB respectively. Mean of observations for BNN over simple condition is also less than: i) SARSA over complex condition by 3.34 dB and ii) SARSA over simple condition by 6.77 dB. Finally, the table shows that the SARSA over complex condition is less than SARSA over simple condition by 3.43. Overall, the Mean Difference and Pukey columns show that SARSA is significantly achieving better PSNR compared to BNN over both simple and complex network conditions.

3.2.3.2 Delay Loss

The packets that arrived at the receiver but after their corresponding *MExT* (mentioned in 2.2.4) are considered as lost packets since they cannot be decoded in time. The average number of such packet losses in each time slot over the entire experiments for SARSA and BNN baseline in simple and complex network conditions are brought in Figure 3.9.

Table 3.4 PSNR Post hoc Tukey's test. It shows the comparison between BNN over complex condition and other interactions. At the last line, the SARSA over complex condition is compared with SARSA over simple condition

Comparison				Mean Difference	SE	df	t	Ptukey
Method	NetType	Method	NetType					
BNN	Complex	BNN	Simple	-32.21	0.217	39,446	-148.7	< .001
		SARSA	Complex	-35.55	0.217	39,446	-164.1	< .001
		SARSA	Simple	-38.98	0.217	39,446	-180.0	< .001
	Simple	SARSA	Complex	-3.34	0.214	39,446	-15.6	< .001
		SARSA	Simple	-6.77	0.214	39,446	-31.7	< .001
SARSA	Complex	SARSA	Simple	-3.43	0.214	39,446	-16.0	< .001

Table 3.5 summarizes that BNN averagely faced larger packet loss due to the delay for both network conditions compared to SARSA. The difference is bigger for complex scenarios than simple. SARSA average faced to even less packet losses due to the delay for complex condition compared to SARSA over simple condition. This shows the effect of the actions selected by SARSA. Figure 3.10 shows the frequency of actions taken by two methods over two network conditions. Table 3.5 and Figure 3.10 show the effect of selecting higher QPs (and smaller bitrate) on facing delays. SARSA faced fewer packet losses for complex compared to simple condition because of its intention to select higher QPs in complex scenarios.

Table 3.5 Mean and Standard deviation of the number of packets considered as lost because of exceeding their tolerable time to be decoded in each time slot for the methods over two network conditions. Note that the loss values must greater than or equal to 0

	Network Condition	Delay Loss
BNN	Simple	0.349 ± 0.479
	Complex	1.8 ± 0.5
SARSA	Simple	0.099 ± 0.294
	Complex	0.066 ± 0.249

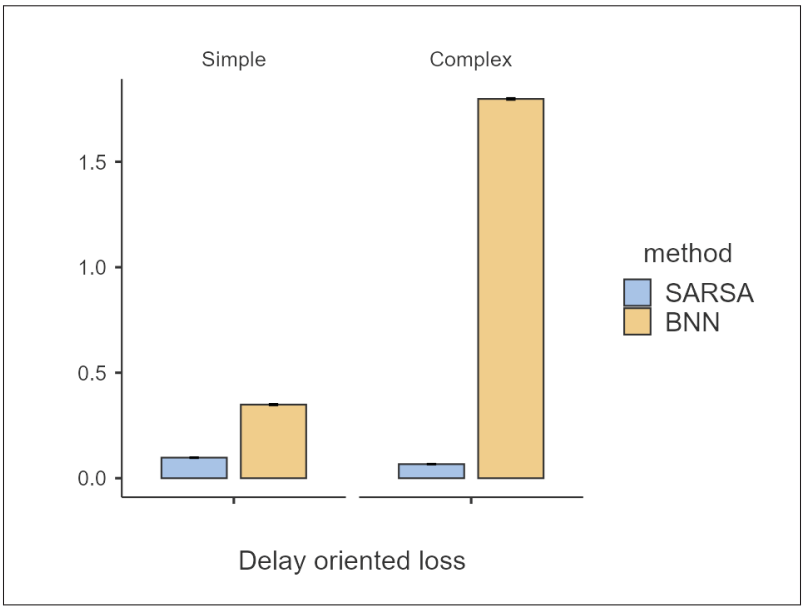


Figure 3.9 The average number of packet loss per time slot due to the delay over the experiments for SARSA (blue) and BNN (yellow) in simple and complex network conditions

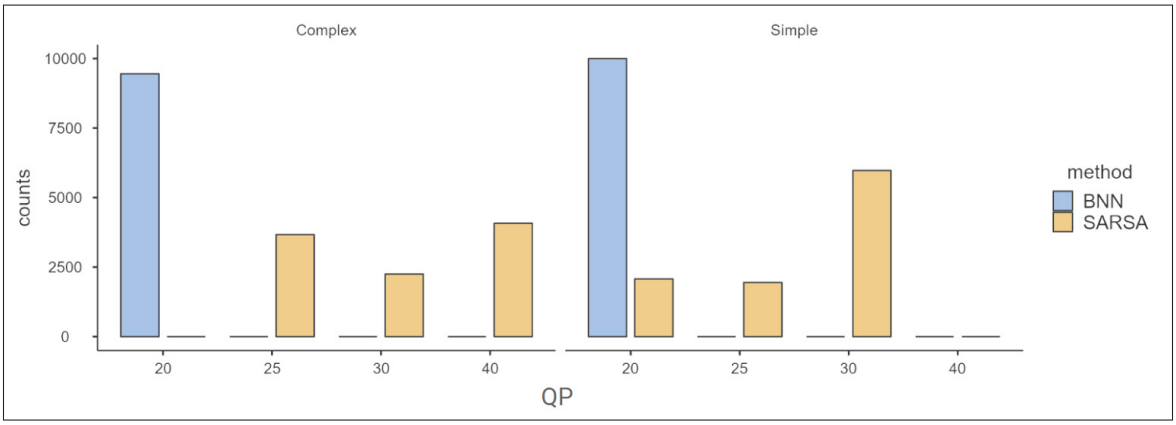


Figure 3.10 Frequency of actions taken in the experiment by both SARSA (yellow) and BNN (blue) methods over simple and complex conditions

Large standard deviation in results of Table 3.5, suggests that maybe the distributions of BNN and SARSA over each network condition are similar. To see the details, we check analysis of the variance (ANOVA).

The difference among the distributions of the results are summarized in Tables of 3.6 and 3.7. The similarity of observations considering a factor or their interaction are analyzed using F or P values. If the F value is large or P value is less than 0.001, it shows that the null hypothesis is not supported for that factor or the interaction. So, it means that there is at least one pair of treatments in that factor or a pair in their interaction that their distributions are not the same. The table shows that each factor and the interaction of them have large F value and p value of less than 0.001. However, we are interested in seeing the details in sets of observations considering the interaction of treatments belonging to two factors, i.e., *method* and *Network type* by doing post hoc test. It lets us see which two sets of observations have different distributions along with comparing the mean difference of each pair.

The *Mean Difference* column of Tukey's post hoc reflects that SARSA had less error compared to BNN in both simple and complex network conditions. We also observe a smaller average packet loss in complex network condition compared to simple network for the SARSA approach. The Ptukey column of all rows contains values less than 0.001, hence, we reject their corresponding null hypothesis which shows that all pairs of the interactions have different distributions. So, it shows that a pair with higher mean value has significantly better result.

Table 3.6 ANOVA on the average number of lost packets due to exceeding their tolerable time to be decoded: Main effect of method and network condition, and their interaction's effect on observed losses. There are two classes for both factors (degree of freedom = 1), and a total 39446 observations. All effects are significant with P < 0.001

	Sum of squares	df	Mean Square	F	P
method	9687	1	9686.642	53,559	< .001
NetType	4957	1	4956.700	27,407	< .001
method * NetType	5397	1	5396.616	29,839	< .001
Residuals	7134	39,446	0.181		

Table 3.7 Post hoc Tukey's test on Loss rate due to exceeding the tolerable time to decode. It shows the comparison between BNN over complex conditions and other interactions. At the last line, the SARSA over complex condition is compared with SARSA over simple condition

Comparison				Mean Difference	SE	df	t	Ptukey
Method	NetType	Method	NetType					
BNN	Complex	BNN	Simple	1.4491	0.00610	39446	237.51	< .001
		SARSA	Complex	1.7313	0.00610	39446	283.76	< .001
		SARSA	Simple	1.7005	0.00610	39446	278.71	< .001
	Simple	SARSA	Complex	0.2822	0.00601	39446	46.92	< .001
		SARSA	Simple	0.2514	0.00601	39446	41.80	< .001
SARSA	Complex	SARSA	Simple	-0.0308	0.00601	39446	-5.12	< .001

3.2.3.3 Link Error Loss

The packets that have not arrived by a maximum waiting time (i.e., $MWT^{j}_{ts_i}$ mentioned in 2.2.4) at each time slot are considered as lost due to link error by the receiver. The average number of such losses in each time slot over the course of entire experiments for SARSA and BNN baseline in simple and complex conditions are depicted on Figure 3.11. As the bit error probability is selected as very low in this experiment, no packet has been lost during the experiment due to real network link error. So, the value that the figure shows for the BNN is for the number of packets that arrived at the receiver but exceeded the maximum waiting time of that time slot. The mean and standard deviation of the this type of lost packets during the experiment for SARSA and BNN over simple and complex scenarios are given in Table 3.8. SARSA shows that all of its packets arrived before the maximum waiting time of the time slot, while BNN faced too long delays for some of its packets. Over complex condition, BNN shows a higher average number of packets considered as being lost due to link error by the receiver compared to BNN over simple. As the means of observations for SARSA and BNN are well separated, we don't use ANOVA to more analysis.

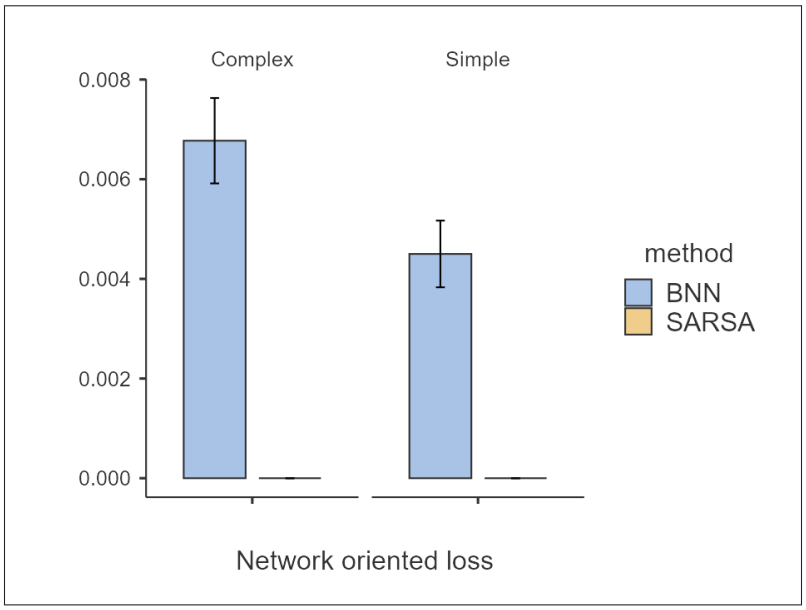


Figure 3.11 The average loss rate due to a network oriented reason over the experiments for SARSA and BNN in simple and complex network conditions

Table 3.8 Mean and Standard deviation of the number of packets considered as loss because of the link error in each time slot of test trajectories for the methods over two network conditions. The loss values are greater than or equal to 0

	Network Condition	Link Error Loss
BNN	Simple	0.004 ± 0.066
	Complex	0.006 ± 0.083
SARSA	Simple	0.0 ± 0.0
	Complex	0.0 ± 0.0

3.2.3.4 Summary

Table 3.9 summarizes the average and standard deviation of all three evaluation measures for both simple and complex network setups.

Table 3.9 Summary of evaluation metrics. The values are greater than or equal to 0

	Network Condition	Delay Loss	Link Error Loss	PSNR (dB)
BNN	Simple	0.349 ± 0.479	0.004 ± 0.066	35.5 ± 0.2
	Complex	1.8 ± 0.5	0.006 ± 0.083	3.27 ± 0.12
SARSA	Simple	0.099 ± 0.294	0.0 ± 0.0	42.2 ± 0.1
	Complex	0.066 ± 0.249	0.0 ± 0.0	38.8 ± 0.1

As reported earlier, SARSA outperforms BNN with significantly better measures (higher for PSNR, lower losses, and lower bitrate consumption) for simple and complex network conditions.

Further analysis of selected actions in each method during the test, confirms that SARSA learned to select a higher variety of actions compared to BNN as illustrated in Figure 3.10. And this action selection properly suits the conditions of both simple and complex networks to optimize the performance.

In more details, in each trajectory, a method can select an action among QPs of 20, 25, 30, 35, 40, and 45. Figure 3.10 illustrates that in complex network conditions, SARSA tends to select higher QP (which leads to lower bit-rate) compared to the simple network conditions.

CONCLUSION AND RECOMMENDATIONS

Conclusion

In this work, we proposed a on-policy 1-step SARSA TD learning method as source rate controller in videoconferencing systems. To the best of our knowledge, we represent the first tabular RL method for real-time interactive video transmission, in particular for videoconferencing. The method learns offline (i.e. RL method is training separately and then is evaluated in (simulated) test environment) by establishing parallel training trajectories over two network conditions (i.e. yield low and high available bitrate as complex and simple scenarios). We evaluated our RL model in these two network conditions and compared it against the Bounded Neural Network (BNN) method which itself had been evaluated with and outperformed some other rule-based rate control algorithms (e.g. GCC and TFRC). We used a video database comprising a set of videos with characteristics similar to those found in videoconferencing applications and with similar characteristics considering several factors such as similarity in the number of people in front of the camera, similar state of camera, etc. We designed a new classical-queuing-model-based network simulator to simply simulate the network status that the videoconferencing flows, transferring from the users located in two subnets, experience. In this work, we leverage two network conditions, with low and high available bitrate, in our experiments.

Running the experiments over such assumptions of videoconferencing database (to consider similar characteristics) and network simulator (to simulate a class of network condition) has two benefits: First, it lets the tabular method to focus to efficiently learn (according to state signal formulation in this work) during the agent's interaction. Second, this work can easily be extended to a generalized experiment by determining some other parameters extracted from video input or encoder/decoder (e.g., Discrete Cosine Transform (DCT) coefficients of each decoded frame and ratio of encoded intra to inter macroblocks as reviewed in Vega, Perra, De Turck & Liotta

(2018)) which capture some intrinsic nature of the video and from the network (e.g., the city, ISP, and server inspired from Sun *et al.* (2016)).

The proposed RL method outperformed the BNN in a sense of providing higher average PSNR, lower losses, and less bitrate consumption in this experiment and shows sophisticate behavior in selecting appropriate actions according to the network conditions to maintain a good quality of experience. Results obtained in this work indicate that the SARSA approach tends to select smaller stream size when it is deployed in complex network condition.

Recommendations

This work paves the way to further research and experiments in real-time communication over the Internet and wireless networks. However, due to the time limitation, we couldn't perform the experiments over other hyper-parameters such as learning rate, discount, temperature parameter in Softmax exploration function, etc. Moreover, we couldn't use the-state-of-the-art network simulator which can well emulate a class of network conditions or run the experiments over the real-world Internet in order to generalize the results.

To extend the work, we have two groups of suggestions. First, to capture some other parameters related to video intrinsic nature and related to the networks which have more correlation to network conditions in order to cluster them. Second, in order to run the experiments over the real-world Internet, the video system needs to be modified carefully. Several considerations need to be taken into account.

First, the encoder should be deployed so that produce continuous a stream after its QP parameter is set. It lets the sender transmit each encoded frame at the time it is encoded (instead of waiting one second to get *fps* encoded frames). In this work, the bad effect of such encoding has been eliminated by pre-storing the encoded version of a videoconferencing file with different QP value).

Second, video traffic coming out of the encoder has a burst nature. This is because of the intra frame which can have large data size compared to the inter frames, especially in the case when the video scene is rather static as in videoconferencing. So, the lack of a pacer unit before transmitting the flow, can lead to video quality flickering which have adverse effect on the QoE (Ni, Eg, Eichhorn, Griwodz & Halvorsen, 2011). The large intra frame size creates a delay that affects the whole communication and requires a special attention.

Third, it is not possible to have the original video to compute the PSNR as a QoE metrics in real-world experiment. Thus, it needs to deploy a no reference visual quality metric to monitor and measure the quality of experience in real time (such as what proposed in Mohamed & Rubino (2002)). It is also recommended to take into account the smoothness of playback, the number and duration of video stallings into the QoE calculation.

Fourth, some RL formulation should be improved to cover the aforementioned changes. State signal should cover more parameters of the environment (e.g., video characteristics and the network conditions). Action signals can be changed such that it leads to smooth playout for the viewer. Reward signal should be changed such that it integrates a new QoE formulation and supports smooth playing.

APPENDIX I

FFMPEG COMMAND

FFmpeg command used in encoding the video:

```
$ ffmpeg -f rawvideo -pix_fmt yuv420p -r 30 -s:v 1280x720 -i
 ↪  pipe:0 -c:v libx264 -force_key_frames
 ↪  expr:gte(t,n_forced*1) -r 30 -profile:v baseline -crf
 ↪  "Agent's_Decision" -f rawvideo pipe:1
```

REFERENCES

Abu-El-Haija, S., Kothari, N., Lee, J., Natsev, P., Toderici, G., Varadarajan, B. & Vijayanarasimhan, S. (2016). Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*.

Alvestrand, H. (2014). Overview: Real time protocols for browser-based applications. *Work in Progress, draft-ietfrtcweb-overview-11*.

Andrade, C. (2020). Understanding the difference between standard deviation and standard error of the mean, and knowing when to use which. *Indian Journal of Psychological Medicine*, 42(4), 409–410.

Aziz, W. A., Qureshi, H. K., Iqbal, A. & Lestas, M. (2020). Accurate Prediction of Streaming Video Traffic in TCP/IP Networks using DPI and Deep Learning. *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 310–315.

Banimelhem, O. & Khasawneh, S. (2012). GMCAR: Grid-based multipath with congestion avoidance routing protocol in wireless sensor networks. *Ad Hoc Networks*, 10(7), 1346–1361.

Blanton, E., Paxson, D. V. & Allman, M. (2009). TCP Congestion Control. RFC Editor. doi: 10.17487/RFC5681.

Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F. & Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1), 1–99.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. & Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1), 1–43.

Carlucci, G., De Cicco, L., Holmer, S. & Mascolo, S. (2016). Analysis and design of the google congestion control for web real-time communication (WebRTC). *Proceedings of the 7th International Conference on Multimedia Systems*, pp. 1–12.

Carlucci, G., De Cicco, L., Holmer, S. & Mascolo, S. (2017). Congestion control for web real-time communication. *IEEE/ACM Transactions on Networking*, 25(5), 2629–2642.

Carofiglio, G., Muscariello, L., Rossi, D. & Valenti, S. (2010). The quest for LEDBAT fairness. *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pp. 1–6.

Chen, Y., Wu, K. & Zhang, Q. (2014). From QoS to QoE: A tutorial on video quality assessment. *IEEE Communications Surveys & Tutorials*, 17(2), 1126–1165.

Cisco, V. N. I. (2019). Cisco visual networking index: Forecast and trends, 2017–2022 white paper. *Cisco Internet Report 17*.

Collins, A. G., Brown, J. K., Gold, J. M., Waltz, J. A. & Frank, M. J. (2014). Working memory contributions to reinforcement learning impairments in schizophrenia. *Journal of Neuroscience*, 34(41), 13747–13756.

De Cicco, L. & Mascolo, S. (2010). A mathematical model of the Skype VoIP congestion control algorithm. *IEEE Transactions on Automatic Control*, 55(3), 790–795.

De Cicco, L., Carlucci, G. & Mascolo, S. (2017). Congestion control for webrtc: Standardization status and open issues. *IEEE Communications Standards Magazine*, 1(2), 22–27.

D'Alconzo, A., Drago, I., Morichetta, A., Mellia, M. & Casas, P. (2019). A survey on big data for network traffic monitoring and analysis. *IEEE Transactions on Network and Service Management*, 16(3), 800–813.

Eggert, L. & Fairhurst, G. (2008). Unicast UDP usage guidelines for application designers. *Internet society*.

Expert, C. C. (2021, Jan, 06). H264 Compression Standard. Consulted at https://www.ccexpert.us/video-conferencing/h264-compression-standard.html.

Fang, J., Ellis, M., Li, B., Liu, S., Hosseinkashi, Y., Revow, M., Sadovnikov, A., Liu, Z., Cheng, P., Ashok, S. et al. (2019). Reinforcement learning for bandwidth estimation and congestion control in real-time communications. *arXiv preprint arXiv:1912.02222*.

Fisher, R. A. (1992). Statistical methods for research workers. In *Breakthroughs in statistics* (pp. 66–70). Springer.

Floyd, S., Handley, M., Padhye, J. & Widmer, J. (2000). Equation-based congestion control for unicast applications. *ACM SIGCOMM Computer Communication Review*, 30(4), 43–56.

Floyd, S., Ramakrishnan, D. K. K. & Black, D. L. (2001). The Addition of Explicit Congestion Notification (ECN) to IP. RFC Editor. doi: 10.17487/RFC3168.

Floyd, S., Handley, M., Padhye, J. & Widmer, J. (2008). TCP friendly rate control (TFRC): Protocol specification.

Fouladi, S., Emmons, J., Orbay, E., Wu, C., Wahby, R. S. & Winstein, K. (2018). Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol. *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pp. 267–282.

Gao, G., Dong, L., Zhang, H., Wen, Y. & Zeng, W. (2019). Content-aware personalised rate adaptation for adaptive streaming via deep video analysis. *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pp. 1–8.

Garcia, J. & Korhonen, T. (2018). Efficient distribution-derived features for high-speed encrypted flow classification. *Proceedings of the 2018 Workshop on Network Meets AI & ML*, pp. 21–27.

Geng, Y., Zhang, X., Niu, T., Zhou, C. & Guo, Z. (2015). Delay-constrained rate control for real-time video streaming over wireless networks. *2015 Visual Communications and Image Processing (VCIP)*, pp. 1–4.

Grieco, L. A. & Mascolo, S. (2004). Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control. *ACM SIGCOMM Computer Communication Review*, 34(2), 25–38.

Guerrero Viveros, M. (2019). Performance Analysis of Google Congestion Control Algorithm for WebRTC.

Ha, S., Rhee, I. & Xu, L. (2008). CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS operating systems review*, 42(5), 64–74.

Hasegawa, G., Kurata, K. & Murata, M. (2000). Analysis and improvement of fairness between TCP Reno and Vegas for deployment of TCP Vegas to the Internet. *Proceedings 2000 International Conference on Network Protocols*, pp. 177–186.

Hayes, D. A. & Armitage, G. (2011). Revisiting TCP congestion control using delay gradients. *International Conference on Research in Networking*, pp. 328–341.

He, Y.-L., Zhang, X.-L., Ao, W. & Huang, J. Z. (2018). Determining the optimal temperature parameter for Softmax function in reinforcement learning. *Applied Soft Computing*, 70, 80–85.

Herglotz, C., Coulombe, S., Vakili, A. & Kaup, A. (2019). Power modeling for virtual reality video playback applications. *2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT)*, pp. 105–110.

Hoe, J. C. (1996). Improving the start-up behavior of a congestion control scheme for TCP. *ACM SIGCOMM Computer Communication Review*, 26(4), 270–280.

Holmer, S., Lundin, H., Carlucci, G., Cicco, L. D. & Mascolo, S. (2016a). *A Google Congestion Control Algorithm for Real-Time Communication* (Report n°draft-ietf-rmcat-gcc-02). Internet Engineering Task Force. Consulted at https://datatracker.ietf.org/doc/html/draft-ietf-rmcat-gcc-02.

Holmer, S., Lundin, H., Carlucci, G., Cicco, L. D. & Mascolo, S. (2016b). *A Google Congestion Control Algorithm for Real-Time Communication* (Report n°draft-ietf-rmcat-gcc-02). Internet Engineering Task Force. Consulted at https://datatracker.ietf.org/doc/html/draft-ietf-rmcat-gcc-02.

Huang, T., Zhang, R.-X., Zhou, C. & Sun, L. (2018a). Delay-Constrained Rate Control for Real-Time Video Streaming with Bounded Neural Network. *Proceedings of the 28th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video*, (NOSSDAV '18), 13–18. doi: 10.1145/3210445.3210446.

Huang, T., Zhang, R.-X., Zhou, C. & Sun, L. (2018b). Qarc: Video quality aware rate control for real-time video streaming based on deep reinforcement learning. *Proceedings of the 26th ACM international conference on Multimedia*, pp. 1208–1216.

Ishimori, A., Farias, F., Cerqueira, E. & Abelém, A. (2013). Control of multiple packet schedulers for improving QoS on OpenFlow/SDN networking. *2013 Second European Workshop on Software Defined Networks*, pp. 81–86.

Jain, R. (1989). A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM SIGCOMM Computer Communication Review*, 19(5), 56–71.

Jansen, J., Cesar, P., Bulterman, D. C., Stevens, T., Kegel, I. & Issing, J. (2011). Enabling composition-based video-conferencing for the home. *IEEE Transactions on Multimedia*, 13(5), 869–881.

Javed, U., Cunha, I., Choffnes, D., Katz-Bassett, E., Anderson, T. & Krishnamurthy, A. (2013). PoiRoot: Investigating the root cause of interdomain path changes. *ACM SIGCOMM Computer Communication Review*, 43(4), 183–194.

Jay, N., Rotman, N. H., Godfrey, P., Schapira, M. & Tamar, A. (2018). Internet congestion control via deep reinforcement learning. *arXiv preprint arXiv:1810.03259*.

Jesup, R. & Sarker, Z. (2021). Congestion Control Requirements for Interactive Real-Time Media. RFC Editor. doi: 10.17487/RFC8836.

Johansson, I. & Sarker, Z. (2017). Self-Clocked Rate Adaptation for Multimedia. RFC Editor. doi: 10.17487/RFC8298.

Karagiannis, G., Chan, K. H., Moncaster, T., Menth, M., Eardley, P. & Briscoe, B. (2012). Overview of Pre-Congestion Notification Encoding. RFC Editor. doi: 10.17487/RFC6627.

Khamassi, M., Velentzas, G., Tsitsimis, T. & Tzafestas, C. (2017). Active Exploration and Parameterized Reinforcement Learning Applied to a Simulated Human-Robot Interaction Task. *2017 First IEEE International Conference on Robotic Computing (IRC)*, pp. 28-35. doi: 10.1109/IRC.2017.33.

Koo, J. & Chung, K. (2010). MARC: Adaptive Rate Control scheme for improving the QoE of streaming services in mobile broadband networks. *2010 10th International Symposium on Communications and Information Technologies*, pp. 105–110.

Kuehlewind, M. (2012). Low extra delay background transport (LEDBAT). *IETF RFC6817*.

Kurdoglu, E., Liu, Y., Wang, Y., Shi, Y., Gu, C. & Lyu, J. (2016). Real-time bandwidth prediction and rate adaptation for video calls over cellular networks. *Proceedings of the 7th International Conference on Multimedia Systems*, pp. 1–11.

Kuzmanovic, A. & Knightly, E. W. (2006). TCP-LP: low-priority service via end-point congestion control. *IEEE/ACM Transactions on Networking*, 14(4), 739–752.

Luo, H. & Shyu, M.-L. (2011). Quality of service provision in mobile multimedia-a survey. *Human-centric computing and information sciences*, 1(1), 1–15.

Luo, H., Shyu, M.-L. & Chen, S.-C. (2008). Video streaming over the internet with optimal bandwidth resource allocation. *Multimedia Tools and Applications*, 40(1), 111–134.

makeinfo. (2021, May, 31). FFmpeg Codecs Documentation. Consulted at https://ffmpeg.org/ffmpeg-codecs.html#libx264_002c-libx264rgb.

Mao, H., Netravali, R. & Alizadeh, M. (2017). Neural adaptive video streaming with pensieve. *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 197–210.

Mohamed, S. & Rubino, G. (2002). A study of real-time packet video quality using random neural networks. *IEEE transactions on circuits and systems for video technology*, 12(12), 1071–1083.

Montgomery, D. C. (2017). *Design and analysis of experiments*. John wiley & sons.

Muhammad, I. & Yan, Z. (2015). SUPERVISED MACHINE LEARNING APPROACHES: A SURVEY. *ICTACT Journal on Soft Computing*, 5(3).

Ni, P., Eg, R., Eichhorn, A., Griwodz, C. & Halvorsen, P. (2011). Flicker effects in adaptive video streaming to handheld devices. *Proceedings of the 19th ACM international conference on Multimedia*, pp. 463–472.

Nihei, K., Yoshida, H., Kai, N., Kanetomo, D. & Satoda, K. (2017). QoE maximizing bitrate control for live video streaming on a mobile uplink. *2017 14th International Conference on Telecommunications (ConTEL)*, pp. 91–98.

Obaidat, M. S., Zarai, F. & Nicopolitidis, P. (2015). *Modeling and simulation of computer networks and systems: Methodologies and applications*. Morgan Kaufmann.

Rejaie, R., Handley, M. & Estrin, D. (1999). RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet. *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, 3, 1337–1345.

Robitza, W. (2017, March, 1). Understanding Rate Control Modes. Consulted at https://slhck.info/video/2017/03/01/rate-control.html.

Saha, N., Bera, S. & Misra, S. (2018). Sway: Traffic-aware QoS routing in software-defined IoT. *IEEE Transactions on Emerging Topics in Computing*, 9(1), 390–401.

Schulzrinne, H., Casner, S. L., Frederick, R. & Jacobson, V. (2003). RTP: A Transport Protocol for Real-Time Applications. RFC Editor. doi: 10.17487/RFC3550.

Shalunov, S., Hazel, G., Iyengar, J., Kuehlewind, M. et al. (2012). Low extra delay background transport (LEDBAT). *RFC 6817*.

Shikari, D. x264 FFmpeg Options Guide. Consulted at https://sites.google.com/site/linuxencoding/x264-ffmpeg-mapping.

Sun, Y., Yin, X., Jiang, J., Sekar, V., Lin, F., Wang, N., Liu, T. & Sinopoli, B. (2016). CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction. *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 272–285.

Sun, Y., Ahmad, I., Li, D. & Zhang, Y.-Q. (2006). Region-based rate control and bit allocation for wireless video transmission. *IEEE Transactions on Multimedia*, 8(1), 1–10.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1), 9–44.

Sutton, R. S. & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., Barto, A. G. et al. (1998). *Introduction to reinforcement learning*. MIT press.

Tokic, M. (2010). Adaptive ε-greedy exploration in reinforcement learning based on value differences. *Annual Conference on Artificial Intelligence*, pp. 203–210.

Tukey, J. W. (1949). Comparing individual means in the analysis of variance. *Biometrics*, 99–114.

Vamplew, P., Dazeley, R. & Foale, C. (2017). Softmax exploration strategies for multiobjective reinforcement learning. *Neurocomputing*, 263, 74–86.

Van der Schaar, M. & Chou, P. A. (2007). *Multimedia over IP and wireless networks: compression, networking, and systems*. Elsevier.

Vega, M. T., Perra, C., De Turck, F. & Liotta, A. (2018). A review of predictive quality of experience management in video streaming services. *IEEE Transactions on Broadcasting*, 64(2), 432–445.

Wassermann, S., Casas, P., Cuvelier, T. & Donnet, B. (2017). NETPerfTrace: Predicting Internet path dynamics and performance with machine learning. *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, pp. 31–36.

Wei, D. X., Jin, C., Low, S. H. & Hegde, S. (2006). FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM transactions on Networking*, 14(6), 1246–1259.

Widmer, J., Boutremans, C. & Boudec, J.-Y. L. (2004). End-to-end congestion control for TCP-friendly flows with variable packet size. *ACM SIGCOMM Computer Communication Review*, 34(2), 137–151.

Wiegand, T. (2003). Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H. 264| ISO/IEC 14496-10 AVC). *JVT-G050*.

Winstein, K., Sivaraman, A. & Balakrishnan, H. (2013). Stochastic forecasts achieve high throughput and low delay over cellular networks. *10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pp. 459–471.

Wu, D., Ci, S. & Wang, H. (2007). Cross-layer optimization for video summary transmission over wireless networks. *IEEE Journal on Selected Areas in Communications*, 25(4), 841–850.

Xu, X., Zuo, L. & Huang, Z. (2014). Reinforcement learning algorithms with function approximation: Recent advances and applications. *Information Sciences*, 261, 1–31.

Zaki, Y., Pötsch, T., Chen, J., Subramanian, L. & Görg, C. (2015). Adaptive congestion control for unpredictable cellular networks. *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pp. 509–522.

Zhang, H., Zhou, A., Lu, J., Ma, R., Hu, Y., Li, C., Zhang, X., Ma, H. & Chen, X. (2020). OnRL: improving mobile video telephony via online reinforcement learning. *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pp. 1–14.

Zhu, J., Vannithamby, R., Rödbro, C., Chen, M. & Andersen, S. V. (2012). Improving QoE for Skype video call in mobile broadband network. *2012 IEEE Global Communications Conference (GLOBECOM)*, pp. 1938–1943.

Zhu, P., Zeng, W. & Li, C. (2007). Joint design of source rate control and QoS-aware congestion control for video streaming over the internet. *IEEE Transactions on Multimedia*, 9(2), 366–376.

Zhu, X., Pan, R., Ramalho, M. A. & de la Cruz, S. M. (2020). Network-Assisted Dynamic Adaptation (NADA): A Unified Congestion Control Scheme for Real-Time Media. RFC Editor. doi: 10.17487/RFC8698.

Zhuo, Y., Wang, J., You, J. & Xue, H. (2017). Video measurement approach with classification based on service performance clustering. *2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN)*, pp. 1240–1244.

Zink, M., Künzel, O., Schmitt, J. & Steinmetz, R. (2003). Subjective impression of variations in layer encoded videos. *International Workshop on Quality of Service*, pp. 137–154.