Learning-TCP A Media-aware Congestion Control Algorithm for Multimedia Transmission

Contents

6.1 Media-aware congestion control formulation 117
6.2 POMDP framework for media-aware congestion control . . 121
6.3 Online Learning 125
6.4 Simulations 128
6.5 Conclusion 131

TCP dominates today's communication protocols at the transport layer in both wireless and wired networks, due to its simple and efficient solutions for end-to-end flow control, congestion control and error control of data transmission over IP networks (see [9] and [11]). However, despite the success of TCP, the existing TCP congestion control is considered unsuitable for delay-sensitive, bandwidth-intense, and loss-tolerant multimedia applications, such as real-time audio streaming and video-conferences (see [9] and [11]). There are two main reasons for this. First, TCP is error-free and trades transmission delay for reliability. Packets may be lost during transport due to network congestion and errors, but TCP keeps retransmitting lost packets until they are successfully transmitted, even if this requires a large delay. The error-free restriction ignores delay deadlines of multimedia packets, i.e. the time by which they must be decoded. Note that even if multimedia packets are successfully received, they are not decodable if they are received after their respective delay deadlines. TCP congestion control adopts an AIMD algorithm. This results in a fluctuating TCP throughput over time, which significantly increases the end-to-end packet transmission delay, and leads to poor performance for multimedia applications [11]. To mitigate these limitations, a plethora of research focused on smoothing the throughput of AIMD-based congestion control for multimedia transmission (see [112] and [113]). These approaches adopt various congestion window updating policies to determine how to adapt the congestion window size to the network congestion. However, these approaches seldom explicitly consider the characteristics of the multimedia applications, such as delay deadlines and distortion impacts of multimedia packets.

In this chapter, we propose a media-aware POMDP-based congestion control, referred to as Learning-TCP, which exhibits an improved performance when transmitting multimedia packets. Unlike the current TCP congestion control protocol that only adapts the congestion window to the network congestion (e.g. the packet loss rate in TCP Reno and the RTT in TCP Vegas), the proposed congestion control algorithm also takes into account multimedia packets' distortion impacts and delay deadlines when adapting its congestion window size. Importantly, the proposed media-aware solution only changes the congestion window updating policy of the TCP protocol at the sender side, without requiring modifications to feedback mechanisms at the receiver.

Note that the multimedia quality obtained by receivers is impacted by the network congestion incurred at bottleneck links, which is only partially observable by senders based on feedback of network congestion signals. In order to capture dynamics of the network congestion and optimize the expected long term quality of multimedia transmissions, we formulate the media-aware congestion control problem using a POMDP framework. The proposed framework allows users to evaluate the network congestion variations over time, and provides the optimal threshold-based congestion window updating policy that maximizes the long-term discounted reward. In this chapter, the considered reward is the multimedia quality, measured using the well-known Peak Signal to Noise Ratio (PSNR).

In practice, the sender needs to learn the network environment during transmission in order to adapt its congestion control policy. Hence, we also propose an online learning approach for solving the POMDP-based congestion control problem. A comparative study of several existing congestion control mechanisms for multimedia applications and the proposed solution is presented in Table 6.1.

Table 6.1: Learning-TCP vs current congestion control solutions for multimedia streaming

Algorithm	**Name of the congestion control**	**TCP-Friendliness**	**Multimedia support**	**Content dependency**	**Decision Type**
Rejaie 1999 [114]	RAP	AIMD-based	Source rate adaptation	No	Myopic
Cai 2005 [112]	GAIMD	AIMD-based	Playback buffering	No	Myopic
Bansal 2001 [113]	Binomial Algorithm	Binomial scheme	Source rate adaptation	No	Myopic
Our approach	Learning-TCP	AIMD-like media aware	Quality-centric congestion control	Yes	Foresighted

This Chapter presents a TCP-like window-based congestion control schemes that use history information, in addition to the current window size and congestion feedback. In summary, this chapter makes the following contributions:

Media-aware congestion control: The proposed Learning-TCP provides a media-aware approach to adapt the AIMD-like congestion control policy to both varying network congestion and multimedia characteristics taking into account source rates, distortion impacts and delay deadlines of multimedia packets. Hence, the media-aware approach leads to a significantly improved multimedia streaming performance.

POMDP-based adaptation: We propose a POMDP framework to formulate the media-aware congestion control problem. It allows the TCP senders to optimize the congestion window updating policy that maximizes the expected long-term quality of multimedia applications. Furthermore, the network user has a partial knowledge about the bottleneck link status. In fact, the number of packets in transit over the bottleneck link queue depends not only on the congestion window of the user, which is known, but also on the congestion windows of all the other users, which cannot be observed. Therefore, the long term prediction and adaptation of the POMDP framework under partial observation of the system state is essential for multimedia streaming, since it can consider, predict, and exploit the dynamic nature of the multimedia traffic and the transmission environment, in order to optimize the application performance.

The POMDP solution is based on a set of updating policies composed of generic congestion control algorithms, with general increase and decrease functions like: AIMD, Inverse Increase/Additive Decrease (IIAD), Square Root inversely proportional Increase/proportional Decrease (SQRT), and Exponential Increase/Multiplicative Decrease (EIMD).

Online learning for delay-sensitive multimedia applications: We present some structural properties of the optimal solution. Thereafter, we propose a practical low-complexity online learning method to solve the POMDP-based congestion control problem on-the-fly. The proposed learning method is designed for multimedia transmission that takes advantage of structural results of the value function.

The chapter is organized as follows. In Section 6.1, we present the media-aware congestion control problem that maximizes the performance of multimedia applications. Thereafter, in Section 6.2, we formulate the problem using a POMDP-based framework. Structural results and the proposed online learning method are presented in Section 6.3. Section 6.4 provides some simulation results that validate the congestion control algorithm, and Section 6.5 concludes the chapter.

6.1 Media-aware congestion control formulation

6.1.1 Network settings

We assume that the network has a set of N end users indexed $\{1, \cdots, N\}$. Each user is composed of a sender node and a receiver node that establish an end-to-end transport layer connection. Let w_n represents the congestion window size of the user n. The network system has some bottleneck links, which results in packet losses when buffers are overloaded. Note that a user cannot observe the traffic generated by other users. In fact, an end user n can only infer the congestion status by observing feedback information from transmitted acknowledgments per RTT. For each acknowledgment, the end user n observes congestion event $o_n \in \{success, fail\}$ (the packet being received successfully or not by the receiver). We consider a time-slotted system with a slot duration of one RTT. Moreover, we assume that the user n has a delay vector $delay_n$ of all packets in its output queue, with $delay_n^i(t+1) = delay_n^i(t) + RTT$ if the i-th packet in the queue is not transmitted during the tth RTT. Before transmitting a packet, the user verifies if $delay_n^i(t) < D_n$, where D_n is the deadline delay of the packet. If not, it drops the packet. The observed information o_n is available to the sender through transmission acknowledgments (ACK) built into the protocol.

6.1.2 Two-level congestion control adaptation

A TCP-like window-based congestion control scheme increases the congestion window after successful transmission of a window of packet, and decreases the congestion window upon the detection of a packet loss event. A general description regarding the congestion

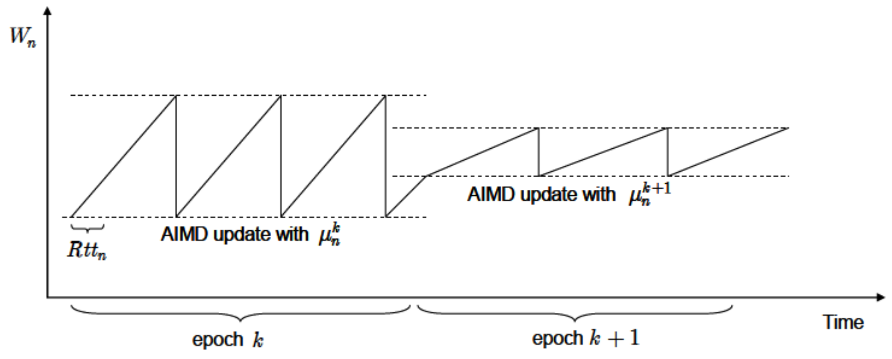

FIGURE 6.1: Congestion window size over time with different update policies per epoch

control window size variation is:

$$w_n \leftarrow \begin{cases} w_n + f(w_n), & \text{if } o_n = success; \\ w_n - g(w_n)w_n, & \text{if } o_n = fail. \end{cases} \tag{6.1}$$

Let us define $\mu_n(w_n) = [f(w_n), g(w_n)] \in \mathcal{A}$, as the updating policy that specifies the two congestion window size variation functions (we refer to $f(w_n)$ as the increasing function and $g(w_n)$ as the decreasing function), where $\mathcal{A}$ represents the set of all updating policies. Some existing examples of updating policies can be found in [112] and [113].

Unlike the existing TCP congestion control that fixes the congestion window updating policy without considering applications' characteristics, the proposed Learning-TCP uses a two-level adaptation to update the congestion window. We define the congestion control epoch $Epoch_n$ as $T \times RTTs$ for user n to periodically change its congestion window updating policy. In fact, we allow the sender to update its policy at the beginning of each epoch, which it cannot change until the next epoch (see Figure 6.1). Indeed, this chapter focuses on how to optimally determine the updating policy, at each epoch, in order to improve the quality of multimedia applications.

6.1.3 Expected multimedia quality per epoch

In this section, we discuss the objective of the proposed media-aware congestion control. Denote application parameters as $\phi_n^k = (R_n^k, D_n^k, A_n^k)$ for user n in the kth epoch, where R_n^k represents the source rate of the multimedia application. The source rate is the average number of packets that arrives at the transmission buffer per second. For example, in a VoIP call, the source rate can be controlled and adapted to the network environment, since there are usually some rate control modules implemented in VoIP software. We further assume an additive distortion reduction function for multimedia applications as in [115], and A_n^k is the additive distortion reduction per packet in epoch

k. A_n^k can be thought of as the media quality improvement of each packet. The following equation depicts the expected distortion reduction per packet for the end user n:

$$E[Q_n^{t,k}(w_n^t, \phi_n^k)] = A_n^k(1 - p_n^k(w_n^t)) \sum_{i=1}^{\min\{w_n^t, buf_n\}} I(delay_n^i(t) \leq D_n^k), \tag{6.2}$$

where buf_n represents the number of packet in the buffer of the user n. The average distortion reduction in the kth epoch is expressed as follows:

$$E[Q_n^k(\mu_n^k, \phi_n^k)] = \frac{1}{T}\sum_{t=1}^{T} E[Q_n^{t,k}(w_n^t, \phi_n^k)]. \tag{6.3}$$

Specifically, a POMDP framework allows users to evaluate the network congestion without perfect knowledge of the overall system state. For each epoch, the proposed Learning-TCP allows the user n to select an optimal updating policy $\mu_n^{opt,k}$ that maximizes the expected distortion reduction in the epoch k, given application parameters ϕ_n^k. Thus, the proposed algorithm performs the following optimization:

$$\mu_n^{opt,k} = \arg\max_{\mu_n^k} \sum_{k=1}^{\infty} \gamma^k E[Q_n^k(\mu_n^k, \phi_n^k)], \tag{6.4}$$

where γ is a discount factor. Note that when the application has no delay deadline, i.e. $D_n^k = \infty$, the objective function in Equation (6.4) is equivalent to maximizing the exponential moving average throughput in the epoch.

During periods of severe congestion, our algorithm may not be TCP-friendly, and therefore penalize other TCP flows. We describe, in the next section, how we adapt our algorithm to be quality-centric and TCP-friendly.

6.1.4 TCP-Friendliness

TCP is not well-suited for emerging multimedia applications because it ignores QoS requirements of the multimedia traffic. To address this issue, some approaches were proposed using end-to-end congestion control schemes [116]. Since TCP is widely used for traffic transport over the Internet, new congestion control schemes should be TCP-Friendly. Therefore, TCP-Friendly congestion control for multimedia has recently become an active research topic (see [117] and [112]). TCP-Friendliness requires that the average throughput of applications using new congestion control schemes does not exceed that of traditional TCP-transported applications under the same circumstances (see [118]). Therefore, we examine the competitive behaviors between TCP and Learning-TCP.

It is well known that the TCP congestion control strategy increases by one or decreases by half the congestion window. Let us consider a scenario with a link having a capacity of r packets per RTT, shared between two flows, one TCP-transported and the other using our media-aware congestion control algorithm.

It is straightforward that updating policies in $\mathcal{A}$ are not necessarily TCP-Friendly (for example, $f(w) = w$ and $g(w) = 1$). However, there exists a non-empty subset of $\mathcal{A}$, whose policies do not violate the TCP-friendliness rule. Proposition 6.1 states that the Learning-TCP algorithm can be TCP-Friendly.

Proposition 6.1. *For all updating policies μ chosen from the set $\mathcal{A}_{fr} = \{\mu(w) = [f(w), g(w)] | f(w) = \frac{3g(w)}{2-g(w)}\}$, the proposed Learning-TCP algorithm is TCP-Friendly.*

Proof. The proof of this proposition is a generalization of the proof of [112] and [119] made for AIMD(α, β). We extend this result for a general updating policies $f(w), g(w) : \mathcal{R} \to \mathcal{R}$. Denote by w_{L-TCP} and w_{TCP} the congestion windows of the Learning-TCP transported flow and the TCP transported flow respectively. Assume that both flows have the same RTT and MSS. The effect due to different RTT and MSS is beyond the scope of this dissertation and is an issue in our future work. On one hand, when $w_{L-TCP} + w_{TCP} < r$, the link is in the underload region and thus, the congestion windows w_{L-TCP} and w_{TCP} evolves as follows:

$$w_{L-TCP}(t + \Delta t) = w_{L-TCP}(t) + f(w_{L-TCP}(t))\Delta t \tag{6.5}$$

$$w_{TCP}(t + \Delta t) = w_{TCP}(t) + \Delta t. \tag{6.6}$$

On the other hand, when $w_{L-TCP} + w_{TCP} \geq r$, the link is overloaded and congestion occurs. We assume that both flows receive the congestion signal once congestion occurs and we denote t_i the ith time that the link is congested. Both flows decrease simultaneously their window based on the following expression:

$$w_{L-TCP}(t_i) + w_{TCP}(t_i) = r \tag{6.7}$$

$$w_{L-TCP}(t_i^+) = w_{L-TCP}(t_i) - g(w_{L-TCP}(t_i))w_{L-TCP}(t_i) \tag{6.8}$$

$$w_{TCP}(t_i^+) = \frac{1}{2}w_{TCP}(t_i). \tag{6.9}$$

The duration between t_i and t_{i+1} is referred to as the ith cycle during which both flows increase their window. Therefore, we have:

$$w_{L-TCP}(t_{i+1}) - w_{L-TCP}(t_i) = -\frac{2g(w_{L-TCP}(t_i)) + f(w_{L-TCP})}{2(f(w_{L-TCP}) + 1))} w_{mL-TCP}(t_i) + \frac{rf(w_{L-TCP})}{2(f(w_{L-TCP}) + 1))} \tag{6.10}$$

Thus, independent of the initial values of w_{L-TCP} and w_{TCP}, after a sufficient number of cycles, the congestion windows of these two flows in the overloaded region converge to:

$$w_{L-TCP}(th) = \frac{f(w_{L-TCP})r}{2g(w_{L-TCP}) + f(w_{L-TCP})}, \tag{6.11}$$

$$w_{TCP}(th) = \frac{2g(w_{L-TCP})r}{2g(w_{L-TCP}) + f(w_{L-TCP})}. \tag{6.12}$$

Therefore, in the steady state, w_{L-TCP} and w_{TCP} increase and decrease periodically. Their average throughput in steady state are expressed by the following:

$$\bar{w}_{L-TCP} = \frac{(2 - g(w_{L-TCP}))f(w_{L-TCP})r}{4g(w_{L-TCP}) + 2f(w_{L-TCP})}, \tag{6.13}$$

$$\bar{w}_{TCP} = \frac{3g(w_{L-TCP})r}{4g(w_{L-TCP}) + 2f(w_{L-TCP})} \tag{6.14}$$

Finally, to guarantee the fairness between the flows, the necessary and sufficient condition is:

$$f(w) = \frac{3g(w)}{2 - g(w)}. \tag{6.15}$$

□

6.2 POMDP framework for media-aware congestion control

In the proposed framework, users have a partial knowledge about the congestion status of bottleneck links. We define the congestion factor C_g, which represents the impact of all users on the congestion status at the bottleneck link. The congestion factor can be seen as a congestion level or occupation level of the bottleneck link. $\mathcal{C}_n$ represents the set of all possible congestion factors. Since the user cannot observe the traffic generated by other users and transmitted over the bottleneck links, it estimates solely the average congestion factor based on history of its observations and actions. Therefore, we formulate the problem with a POMDP framework. Moreover, the objective function to optimize can be rewritten as follows:

$$U_n = \sum_k \gamma^k \sum_{t=1}^{T} A_n^k (1 - p_n^k(w_n^t)) \sum_{i=1}^{\min\{w_n^t, buf_n\}} I(delay_n^i(t) \leq D_n^k). \tag{6.16}$$

Note that the end user tries to maximize the number of packets successfully transmitted before their delay deadlines.

6.2.1 POMDP-based congestion control

Based on Equation (6.16), we define a POMDP-based congestion control of user n as a follows:

Action: The user selects the congestion window updating policy $\mu_n^k \in \mathcal{A}$, where μ_n^k is the updating policy of user n in the kth epoch.

State: The state is defined as $X_n^k = \{C_g, \phi_n^k\} \in \mathcal{X}_n$. The application parameters ϕ_n^k are known by the user n. However, the congestion factor $C_g \in \mathcal{C}_n$, which is impacted by the overall traffic transiting in the bottleneck link, cannot be directly observed by the users. The user n has to infer the congestion factor based on the observed information and feedback.

At each time slot, the system has a congestion factor C_g. The user takes an action μ_n, which causes the environment to transit to C_g' with probability $T(C_g', \mu_n, C_g)$. Having the congestion factor C_g', the user observes o_n with probability $O(o_n, C_g', \mu_n)$. The belief about the congestion factor is defined as $b : \mathcal{C}_n \to [0, 1]$. The function $b(.)$ represents the probability distribution of the congestion factor at the kth epoch. Denote the chosen congestion factor (i.e., inferred by the end user as the most likely of all possible congestion factors) at the kth epoch by C_g^k. The belief distribution of the congestion factor $b(C_g)$ is updated as follows:

$$\begin{aligned} b_n^k(C_g') &= \frac{Pr(o_n|C_g', \mu_n^k, b)Pr(C_g'|\mu_n^k, b)}{Pr(o_n|\mu_n^k, b)}; \\ &= \frac{O(o_n, C_g', \mu_n^k) \sum\limits_{C_g \in \mathcal{C}_n} T(C_g', \mu_n^k, C_g) b_n^{k-1}(C_g)}{Pr(o_n|\mu_n^k, b)}. \end{aligned} \tag{6.17}$$

The denominator, $Pr(o_n|\mu_n, b)$, can be treated as a normalizing factor, independent of C_g' that causes b to sum to 1.

The probability $p_n^k(w_n)$ represents the average packet loss rate in the kth epoch when the congestion window size is w_n, which can be calculated as follows:

$$p_n^k(w_n) = \sum_{C_g \in \mathcal{C}_n} Prob(C_g \geq \widetilde{C}_g|w_n) b_n(C_g), \tag{6.18}$$

where $\widetilde{C}_g$ is the congestion level at the bottleneck link, which is not observable by users. However, the average packet loss rate itself is observable by users, given a certain congestion window w_n.

Utility: Based on Equation (6.16), the utility of user n is defined as the discounted long-term expected reward:

$$U_n = \sum_{k=1}^{\infty} \gamma^k \sum_{C_g \in \mathcal{C}_n} u_n(X_n^k, \mu_n^k) b(C_g), \tag{6.19}$$

where $u_n(X_n^k, \mu_n^k) = \sum_{t=1}^{T} A_n^k (1 - p_n^k(w_n^t)) \sum_{i=1}^{\min\{w_n^t, buf_n\}} I(delay_n^i(t) \leq D_n^k)$ represents the immediate reward in the kth epoch.

A policy $\mu_n^{opt} = \{\mu_n^{opt,1}, \mu_n^{opt,2}, ...\}$ that maximizes U_n is called an optimal policy that specifies for each epoch k the optimal updating policy $\mu_n^{opt,k}$ to use. The optimal value function U_n^k satisfies the following Bellman equation:

$$U_n^k(C_g^k) = \max_{\mu_n^k \in \mathcal{A}} \{u_n(X_n^k, \mu_n^k) + \gamma \sum_{C_g' \in \mathcal{C}_n} T(C_g' | \mu_n^k, C_g) U_n^{k+1}(C_g')\}. \tag{6.20}$$

The optimal policy at the kth epoch is expressed as follows:

$$\mu_n^{opt,k} = \arg \max_{\mu_n^k \in \mathcal{A}} \{u_n(X_n^k, \mu_n^k) + \gamma \sum_{C_g' \in \mathcal{C}_n} T(C_g' | \mu_n^k, C_g) U_n^{k+1}(C_g')\}. \tag{6.21}$$

We prove in the next section the existence of optimal stationary policy and we show how to determine such policy for our POMDP problem.

6.2.2 Existence of optimal stationary policy

Because of the difficulty of computation and implementation of the optimal solution for POMDP-based problems, we would like to restrict attention to stationary policies when seeking optimal solution. Note that we formulate our problem as an infinite horizon POMDP with expected discounted reward.

The belief set is continuous, which may lead to an explosion of the solution size and the computation complexity. Therefore, we transform the belief set to a discrete set. We use an aggregation function that maps the belief states into a discrete set of beliefs. An example of aggregation function is presented in Section 6.3. Moreover, for each belief, we assume that there is a finite set of actions $\mathcal{A}$. Under these assumptions, Theorem 6.2.10 of [92] can be applied and we can prove the existence of an optimal stationary policy for our POMDP problem. Therefore, we restrict our problem to the set of stationary policies. We are able to determine an algorithm that computes one such policy. We can now omit the epoch index k, as the optimal stationary policies depend only on ϕ and C_g. The goal of this POMDP problem is therefore to find a sequence of updating policies μ_n that maximizes the expected reward. For each belief, the value function can be formulated as follows:

$$U_n(C_g) = \max_{\mu_n \in \mathcal{A}} \{u_n(X_n, \mu_n) + \gamma \sum_{C'_g \in \mathcal{C}_n} T(C'_g|\mu_n^k, C_g) U_n(C'_g)\} \tag{6.22}$$

Specifically, a powerful result of [34] and [35] says that the optimal value function for our POMDP problem is PWLC in the belief. Then, every value function can be represented by a set of hyper-planes denoted Υ-vectors, Γ_k, where $U_n(C_g) = \max_{\Upsilon \in \Gamma_k} b(C_g)\Upsilon$. Γ_k is updated using the value iteration algorithm through the following sequence of operations:

$$\Gamma_{k+1}^{\mu,o_n} \leftarrow \Upsilon_\mu^{o_n}(X_n) = \frac{u_n(X_n,\mu)}{|o_n|} + \gamma \sum_{X' \in \mathcal{X}} T(X_n, \mu, X') O(o_n, C'_g, \mu) \Upsilon(X'), \forall \Upsilon \in \Gamma_k, \tag{6.23}$$

$$\Gamma_{k+1}^{\mu} = \oplus_{o_n} \Gamma_{k+1}^{\mu,o_n}; \tag{6.24}$$

$$\Gamma_{k+1} = \cup_{\mu \in \mathcal{A}} \Gamma_{k+1}^{\mu}. \tag{6.25}$$

Note that each Υ-vector is associated with an action that defines the best updating policies for the previous $(k-1)$ epochs. The kth horizon value function can be expressed as follows:

$$U(C_g) = \max_{\mu_n \in \mathcal{A}} \left[u_n(X_n^k, \mu_n) + \gamma \sum_{o_n} \max_{\Upsilon \in \Gamma_k^{\mu_n, o}} \sum_{C'_g \in \mathcal{C}_n} P_n(C'_g|C_g) O(o_n, C'_g, \mu) \Upsilon \right]. \tag{6.26}$$

Many algorithms were proposed to implement solutions for POMDP problems by manipulating the Υ-vector using a combination of set projection and pruning operations (see [34],[95] and [120]).

The main difficulty of POMDP-based optimization is the prohibitively high computational complexity and the assumption that statistics, such as the state transition probability are priory known, which may be not true in practice. To overcome this obstacle, we propose an online learning method that allows the sender to determine the optimal congestion control policy on-the-fly, with a low computational complexity.

6.3 Online Learning

Solving a POMDP is an extremely difficult computational problem. In this section, we show how the value function can be updated on-the-fly, and with a low computation complexity, in order to solve the POMDP problem described in the previous section. In the proposed learning model, the user maintains the state-value function $Q(\mu_n, \phi, C_g)$ as a lookup table, which determines the optimal policy in the current slot. In fact, the state-value function $Q(\mu_n, \phi, C_g)$ is updated as follows:

$$Q(\mu_n^{k-1}, \phi^{k-1}, C_g^{k-1}) \leftarrow \beta_k Q(\mu_n^{k-1}, \phi^{k-1}, C_g^{k-1}) + (1-\beta_k)(U_n + \gamma Q(\mu_n^k, \phi^k, C_g^k)), \quad (6.27)$$

where β_k is a learning rate factor satisfying $\sum_{k=1}^{\infty} \beta_k = \infty, \sum_{k=1}^{\infty} (\beta_k)^2 < \infty, e.g. \beta_k = \frac{1}{k}$. At the epoch $k-1$, the user gets the application parameters ϕ^{k-1}, estimates the congestion factor C_g^{k-1}, and chooses the policy μ_n^{k-1} that maximizes $Q(\mu_n^{k-1}, \phi^{k-1}, C_g^{k-1})$. At the epoch k, the user obtains the new application parameters, estimates the congestion factor, chooses a congestion window updating policy, and updates the state-value function $Q(\mu_n^{k-1}, \phi^{k-1}, C_g^{k-1})$.

The large state space $\mathcal{X}_n$, due to the continuous space of congestion factors, may prohibit an efficient learning solution, due to the complexity and the long convergence time. We propose to adopt an effective state aggregation mechanism to reduce the complexity and the convergence time of the learning algorithm. As an example of the aggregation function, we may quantize the congestion factor to the nearest integer.

6.3.1 Adaptive state aggregation

We propose to use an aggregation function that maps the congestion factor space $\mathcal{C}_n$ into a discrete space, as we have assumed in Section 6.2.2. This function aggregates the adjacent average congestion factors $C'_g \in \tau_n \subset \mathcal{C}_n$ into a representative average congestion factor value C_g. In this chapter, we propose an adaptive state aggregation method that iteratively adapts the aggregation function. Let $\Delta(C_g, U_n^k, \delta)$ represent the adaptive aggregation function, defined as follows:

$$\Delta(C_g, U_n^k, \delta) = C_g^n = \frac{C^L + C^H}{2}, \quad (6.28)$$

where $C^L = invU_n^k(U^{min} + (l-1)\delta)$, $C^H = invU_n^k(U^{min} + l\delta)$, and $(l-1)\delta \leq U_n^k(w|C_g) - U^{min} < l\delta$. Note that $invU_n^k$ represents the inverse function of $U_n^k(w|C_g)$, U^{min} denotes the minimum value of the expected utility of the user starting from the previous epoch, and δ is referred to as the utility spacing that determines the aggregation function from the expected utility-to-go domain.

6.3.2 Structural Properties

In this section, we develop some structural properties of the optimal policy and corresponding value function, based on which we will then discuss approximation results of the value function. This approximation allows us to represent compactly the value function. It was proved, in [35], that the optimal value function U_n^* is PWLC with respect to the belief vector. As we are considering a discrete set of average congestion factors, the value function can be approximated using a PWLC function. Importantly, we are able to control the computational complexity and achievable performance by using different predetermined approximation error thresholds δ.

Algorithm 2 Online learning algorithm for POMDP-based congestion control

```
Initialize $Q(\mu_n^k, \phi_n^k, C_g) = 0$ for all possible application parameters, congestion factor and updating policy;
Initialize $\phi$, $\mu_n$ and $C_g$;
$U_n = 0$;
while true do
    $\phi^{prev} = \phi$;
    $\mu_n^{prev} = \mu_n$;
    $C_g^{prev} = C_g$;
    Get the new application parameters $\phi$;
    Select the policy and congestion factor such as: $(\mu_n^k, C_g) = \arg\max_{\mu_n, C_g} Q(\mu_n, \phi, C_g) b_n(C_g)$ with probability $(1 - \epsilon)$, else choose a random policy and congestion factor;
    $Q(\mu_n^{prev}, \phi^{prev}, C_g^{prev}) \leftarrow \beta_k Q(\mu_n^{prev}, \phi^{prev}, C_g^{prev}) + (1 - \beta_k)(U_n + \gamma Q(\mu_n, \phi, C_g))$;
    $U_n = 0$;
    for $t = 1 \to T$ do
        Transmit packets using the updating policy $\mu_n$ and the congestion factor $C_g$;
        Update the congestion window based on Equation (6.1);
        $U_n = U_n + A_n^k \times recPkt$, where $recPkt$ is the number of packets received before their delay deadlines.
    end for
    Update the beliefs based on Equation (6.17);
end while
```

We propose, in this section, a low-complexity online learning algorithm based on an extension of the on-policy TD-λ Algorithm [121], described in Algorithm 3. The proposed learning method is greatly impacted by the utility spacing δ, and the number of states in an epoch depends on the aggregation function $\Delta(C_g, U_n^k, \delta)$. The size of the average congestion set in the kth epoch is $\lceil \frac{U^{k,max} - U^{k,min}}{\delta} \rceil + 1$.

At the beginning of epoch k, the user receives the application parameters ϕ_n^k from the upper layer, and selects the updating policy and the congestion factor that maximize its state-value function. Then, the user transmits its packets during the epoch using the chosen policy. At the end of the epoch, the user updates the state-value function based on observation during the epoch. The following lemma proves the convergence of the proposed algorithm.

Lemma 6.2. *The proposed learning algorithm converges to the optimal value function w.p.1.*

Proof. The proof of this lemma follows from the Theorem 1 of [122]. In fact, Sarsa algorithm converges to the optimal values function whenever the following assumptions hold:

1. The state space and the action space are finite,
2. β_k satisfies $\sum_{k=1}^{\infty} \beta_k = \infty, \sum_{k=1}^{\infty} (\beta_k)^2 < \infty, e.g. \beta_k = \frac{1}{k}$,
3. The reward function is bounded.

It is straightforward that the previous assumptions hold for our problem, and therefore, the Algorithm 3 converges to the optimal values function. □

6.3.3 Implementation and complexity

Although value iteration algorithms give an exact solution of POMDP optimization problems, those algorithms require a time and space complexity that may be prohibitively expensive. In fact, to better understand the complexity of exactly solving the POMDP problem, let $|\Gamma_k|$ be the number of Υ-vectors in the kth epoch. In the worst case, the Υ-vectors size in the $(k+1)$-th epoch is $|\mathcal{A}| \times |\Gamma_k|$ (see [123]), and the running time will be $|\mathcal{X}_n|^2 \times |\mathcal{A}| \times |\Gamma_k|$. It also requires solving a number of LPs for pruning vectors.

Interestingly, the proposed algorithm has a state space of $|\mathcal{A}| \times |\mathcal{C}_n| \times |\Phi|$, and has a polynomial time complexity. Therefore, this algorithm can be implemented on mobile devices as it takes only a polynomial time when seeking for the optimal policy. Moreover, the proposed algorithm is implemented only at the transmitter side and is transparent for the receiver. We do not even require any change at routers. Moreover, as we have proved that Learning-TCP is TCP-Friendly, any other congestion control algorithm can be implemented in parallel. For first epochs, the Learning-TCP algorithm may give suboptimal performance. However, a near-optimal result can be obtained after a sufficient number of epochs. Interestingly, we can significantly speed up the learning and avoid this problem if the state-value functions are initialized with the values obtained the last time Learning-TCP was used.

6.4 Simulations

In this section, we present some simulation results using MATLAB-based simulations of the proposed Learning-TCP algorithm. Note that we do not study the performance of congestion control schemes (AIMD, Binomial,...) as they were already deeply investigated. Instead, we analyze the performance of LearningTCP that chooses one congestion control schema every epoch. We consider that multimedia users are transmitting video sequences at a variable bit rate of $\mathcal{R} = \{1, 1.25, 1.5, ..., 5.75, 6\}$ Mbps. We assume that packets can tolerate a delay of $\mathcal{D} = \{133, 266, 400, ..., 800\}$ ms, and we set the packet length to 1024 Bytes. Moreover, we assume that each frame has an additive distortion per packet in the set $\mathcal{A}_{distor} = \{0.05, 0.06, ..., 0.16\}$. We consider also a set of policies $\mathcal{A}$ composed of IIAD and SQRT policies defined as follows:

$$\text{IIAD: } f(w) = \frac{3\beta}{2w - \beta} \text{ and } g(w) = \frac{\beta}{w}; \tag{6.29}$$

$$\text{SQRT: } f(w) = \frac{3\beta}{2\sqrt{w+1} - \beta} \text{ and } g(w) = \frac{\beta}{\sqrt{w+1}}; \tag{6.30}$$

where $\beta \in \{0.1, 0.2, ..., 0.9\}$. We consider the set of average congestion factors $\mathcal{C}_n = \{1, 2, .., 50\}$, and we set γ to 0.1.

6.4.1 TCP-fairness

We focus, first, on the fairness of our proposed Learning-TCP. Figure 6.2 shows how the proposed algorithm interacts with TCP transported flows depending on QoS parameters chosen from the set $\Phi = \mathcal{R} \times \mathcal{D} \times \mathcal{A}_{distor}$. In order to study this effect, we simulate 10 connections: 5 with TCP and 5 connections using the Learning-TCP algorithm, within different QoS requirements and application parameters. We illustrate, in Figure 6.2, the fairness ratio depending on the delay deadline and source rate. The fairness ratio (see [113] and [114]) is defined by the ratio between the total throughput of Learning-TCP connections and total throughput of TCP connections. The closer the fairness ratio is to 1, the friendlier will the congestion control be to other TCP flows. We observe that Learning-TCP has a fairness ratio close to 1 except with hard deadline delay and high source rate. In fact, as we can see in Figure 6.2, when the delay deadline is lower than 300 ms and the source rate is higher than 4 Mbps, the fairness ratio is between 1.2 and

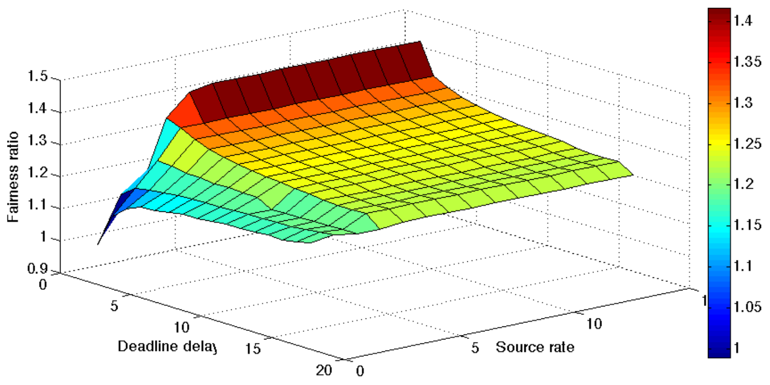

FIGURE 6.2: Fairness ratio of Learning-TCP for different source rates and delay deadlines .

1.45. Indeed, with hard deadline delays and high source rates, the user needs higher throughput in order to satisfy its QoS requirement.

6.4.2 Learning-TCP algorithms and fixed-policy algorithms

In this section, we investigate the interactions between Learning-TCP and other multimedia congestion control algorithms. We consider a bottleneck link of capacity 100 Mbps shared between 10 users (one Learning-TCP; one TCP and 8 users using Binomial congestion control) as described in Table 6.2. We simulate a video transmission application during 350 time slots, and we assume that users receive a new set of application parameters every epoch, T=50 RTT, where the RTT duration is 100 ms. In order to illustrate the impact of the delay on the congestion control algorithms, we assume that the deadline delay is 133 ms in the first epoch, and that it increases by 133 ms every epoch. A real use-case of these simulation settings is a streaming application, where the user may change the required quality at each epoch. Let us consider a congested network, the user decreases at the end of each epoch the required quality of the streaming, and increases the deadline delay, thereby decreases the packet loss probability. We observe, in Figure 6.3, that the Learning-TCP uses different policies for each delay deadline. For hard delay deadlines, we observe that the throughput of the Learning-TCP user is higher than the throughput of other users. Figure 6.5 illustrates the throughput of TCP user and Figure 6.4 illustrates the throughput of Binomial congestion control users. The Binomial-CC users obtain the highest average throughput (9.2 Mbps Versus 7.65 Mbps for TCP and 8.36 Mbps for Learning TCP). In fact, as we can see in Figure 6.3, the Learning-TCP gives the highest throughput for hard delay deadlines. However, it is still TCP-friendlier in the average. Interestingly, we show in the next section, how the proposed algorithm gives better video quality when obeying the friendliness rule.

Table 6.2: Users in the network

	IIAD1	IIAD2	IIAD3	IIAD4	TCP
I	$\frac{0.3}{w-0.1}$	$\frac{0.6}{w-0.2}$	$\frac{0.9}{w-0.3}$	$\frac{1.2}{w-0.4}$	1
D	$\frac{0.2}{w}$	$\frac{0.4}{w}$	$\frac{0.4}{w}$	$\frac{0.8}{w}$	0.5
	SQRT1	SQRT2	SQRT3	SQRT4	LEARNING-TCP
I	$\frac{3}{8\sqrt{w+1}-1}$	$\frac{3}{4\sqrt{w+1}-1}$	$\frac{9}{8\sqrt{w+1}-3}$	$\frac{3}{2\sqrt{w+1}-1}$	f(w)
D	$\frac{0.25}{\sqrt{w+1}}$	$\frac{0.5}{\sqrt{w+1}}$	$\frac{0.75}{\sqrt{w+1}}$	$\frac{1}{\sqrt{w+1}}$	g(w)

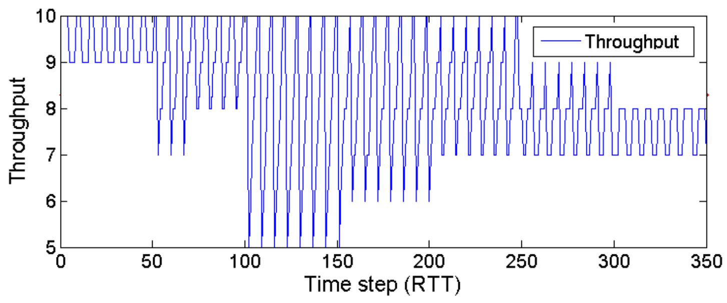

Figure 6.3: Throughput of Learning-TCP.

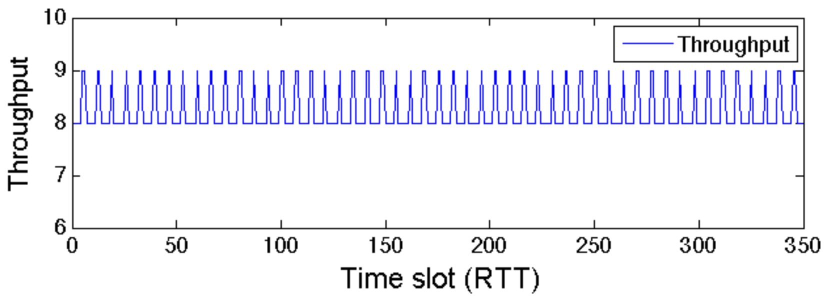

Figure 6.4: Throughput of Binomial-CC.

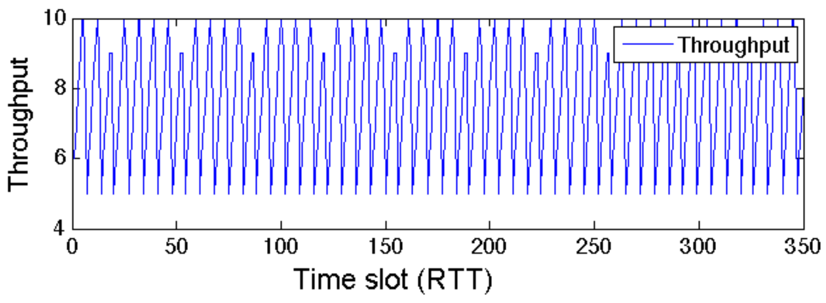

Figure 6.5: Throughput of TCP.

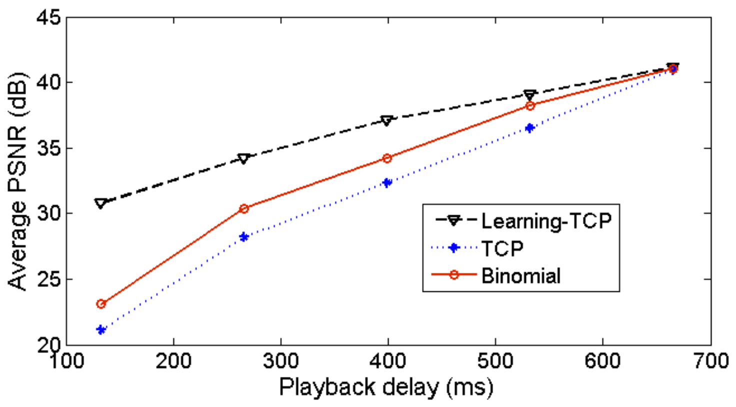

FIGURE 6.6: Average received video quality using different congestion control for multimedia transmission.

6.4.3 Performances of Learning-TCP against others multimedia congestion control algorithms

In order to evaluate the video quality (measured through the average PSNR, in decibels) using different congestion control algorithms, we consider the previous scenario where 4 users use Binomial congestion control algorithm; 4 users use TCP and two users using a Learning-TCP algorithm.

We simulate the transmission of a video sequence with length of 50 s (CIF resolution, 50 Hz frame-rate) and compressed by an H.264/AVC codec (any codec can be used, we used this one just for illustrative purposes). We assume that users receive different values of source rate and additive distortion per packet at every epoch. The delay deadline varies between 133 ms and 800 ms. Figure 6.6 illustrates the video quality obtained with different congestion control algorithms. We observe that the Learning-TCP leads to better video quality. Therefore, our proposed approach outperforms others, especially for real-time applications with hard deadline delay such as video-conferencing applications for example. In fact, as illustrated in Figure 6.7, Learning-TCP users obtain the highest percentage of packets delivered before their delay deadline. Indeed, our algorithm is able to optimize the congestion window by considering the distortion impact, delay deadline and the source rate.

6.5 Conclusion

In this chapter, we have formulated the media-aware congestion control problem as a POMDP that considers the distortion impact, delay deadline and the multimedia source rate. We have considered a set of generic TCP-friendly updating functions for the

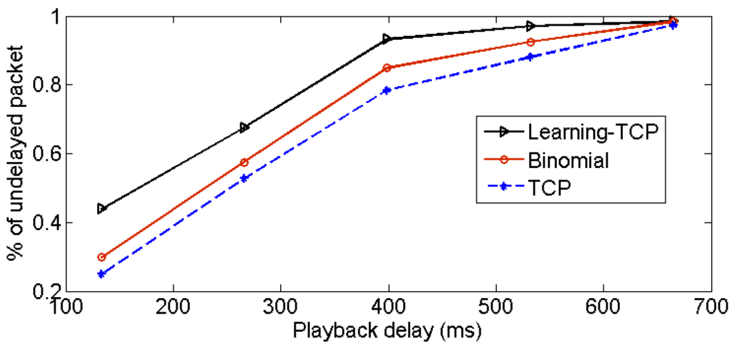

FIGURE 6.7: The percentage of packets delivered before their delay deadlines.

congestion window adaptation. The optimal policy allows the sender to optimize the congestion window size that maximizes the long-term expected quality of the multimedia application. We have also proposed an online learning method to solve the Learning-TCP on-the-fly. Simulation results have shown that the proposed congestion control algorithm outperforms conventional TCP-friendly congestion control schemes in terms of quality, especially for real-time applications with hard delay deadlines. Moreover, the proposed Learning-TCP algorithm is implemented only at the sender side, and is transparent to the routers and the receiver.

Note that we have considered only the impact of QoS parameters (delay, source rate, etc.) on the congestion control. In the next chapter, we focus on the quality perceived by end users through a QoE-based congestion control algorithm. In fact, we consider that users maximize the QoE, based on MOS feedbacks from receivers.