R-FEC: RL-based FEC Adjustment for Better QoE in WebRTC

Abstract

视频会议应用的需求呈爆炸式增长，但用户仍然经常面临体验质量(QoE)不理想的问题。视频会议应用采用前向纠错(Forward Error Correction, FEC)作为恢复机制，以满足严格的时延要求，克服网络中普遍存在的丢包问题。然而，许多研究主要集中在视频速率控制上，而忽略了这种视频恢复机制对速率控制的复杂交互作用及其对用户QoE的影响。在动态变化的网络环境下，为当前的视频速率确定合适的FEC量并不是一件简单的事情。例如，较高的FEC可能会增强对丢包的容忍度，但由于FEC处理开销可能会增加延迟，并且由于用于FEC的额外带宽而损害视频质量。为了解决这个问题，我们提出了R-FEC，这是一个基于强制学习(RL)的框架，用于视频会议中的视频和FEC比特率决策。R-FEC旨在通过自动学习过去决策的结果，并调整视频和FEC比特率，以最大限度地提高用户QoE，同时最大限度地减少网络拥塞，从而提高整体QoE。我们的实验表明，R-FEC在视频会议中优于最先进的解决方案，与默认的WebRTC相比，其视频速率提高了27%，视频质量提高了6dB PSNR。

1. Introduction

对视频会议的需求比以往任何时候都呈指数级增长。根据Gartner的研究[8]，到2022年，全球31%的员工预计将成为远程工作者。随着越来越多的人远程工作和互动，视频会议作为一项关键的使能技术正受到越来越多的关注。然而，众所周知，用户偶尔会面临低质量的体验(QoE)，如屏幕模糊和死机，因为互联网的最佳努力数据包传输无法保证视频会议中用户的QoE。在这方面，视频会议应用依赖于他们的比特率控制算法来适应不同的网络条件，类似于视频流中的自适应比特率(ABR)算法，但由于其更严格的延迟预算，更具挑战性。因此，如何设计一种有效的视频会议速率控制算法一直是一个有待解决的问题。

速率控制算法的传统方法是使用具有固定阈值的预定义控制规则[17]。然而，这种基于规则的设计需要大量的工作来调优这些阈值以获得良好的性能。此外，由于具有固定阈值的非自适应控制规则，往往不能适应时变的网络条件。近年来，人们探索了基于机器学习的方法来解决这个问题。例如，Pensieve[18]使用强化学习(RL)来提高视频流中的QoE。同样，Concerto[35]采用模仿学习来协调其视频编解码器和传输层度量，以减轻屏幕冻结的问题。另一方面，OnRL[34]利用在线RL来控制移动视频会议应用的比特率。

虽然显示出一些潜力，但现有的基于机器学习的方法只关注在当前网络条件下确定合适的视频比特率，而在很大程度上忽略了损失恢复机制的复杂相互作用。由于即使在网络不严重拥塞的情况下也可能偶尔发生丢包，因此视频会议应用程序需要立即恢复丢包，以满足实时通信的严格延迟要求。例如，FEC (Forward Error Correction)是指通过FEC报文的冗余，主动恢复丢失的媒体报文。

然而，确定FEC的最佳量是一个重大挑战，因为FEC如果被滥用也会增加延迟或损害视频质量。更具体地说，FEC适用于在丢包情况下恢复数据包，而不会产生额外的数据包恢复往返时间。但如果过度使用，则会显著降低用户QoE，因为较大的FEC部分会大大降低有效视频速率，从而导致视频质量差。此外，它还会增加FEC处理开销带来的额外延迟。

从我们的关键观察(详见§3)中，我们发现WebRTC，视频会议的事实上的标准，过度使用FEC来克服数据包丢失。图1显示了三种不同FEC比例级别的WebRTC的行为:禁用FEC (disabled)、默认WebRTC使用的FEC方案(default)和调优FEC (tuned)。我们修改默认的WebRTC以纳入我们的更改，并使用4G带宽跟踪模拟现实的4G网络[27]。

它们的吞吐量非常相似，因为它们使用相同的比特率控制算法。在根据可用带宽控制其吞吐量的同时，它们会在大约30秒的间隔内经历数据包丢失，这在移动4G通道中很常见。我们还观察到流延迟的峰值显然，最糟糕的情况是没有FEC的WebRTC，因为解码器等待丢失帧的丢失数据包的重传。由于图1 (a)中显示的延迟增加，用户会出现抖动或冻结屏幕，这也表现为图1 (b)中较差的QoE。另一方面，调整FEC的WebRTC表现出比其他WebRTC更好的延

迟控制，这表明我们可以通过调整FEC比例来获得更好的性能并提高QoE。我们还观察到，即使传输延迟变化不显著，当传输延迟波动时，WebRTC也会误判动态网络条件。这种定量分析激发了我们对以下问题的研究:我们能否为视频和FEC比特率决策开发一种RL方法，这种方法可以学习和适应网络条件，从而在没有任何预编程规则和阈值的情况下最大化用户QoE？

在本文中，我们提出了使用RL模型来适应视频和FEC速率的R-FEC，以取代WebRTC中默认的FEC控制器和Google拥塞控制(GCC)算法。在观察到网络拥塞和FEC误用的情况下延迟会增加的基础上，我们对RL模型的设计进行了改进。因此，我们将跟踪延迟变化作为设计RL模型的关键因素。我们的强化学习算法不断从过去的行为中学习，将当前网络状态映射到视频编码和FEC的目标比特率，而无需任何预编程规则。我们让RL算法学习如何在动态网络条件下避免视频和FEC速率的过度使用或使用不足。
综上所述，我们有以下贡献:

•我们分析了WebRTC的性能，WebRTC是所有主要浏览器中用于视频会议的开放web标准。然后，我们调查了导致性能差的主要原因，并使用机器学习方法解决了这些问题。

•据我们所知，这是第一次利用RL方法来决定视频和FEC速率，以提高视频会议的QoE。RL方法允许系统跟踪网络状况变化，并在丢包情况下优化视频和FEC比特率。

•我们在WebRTC框架之上实现R-FEC，并验证其在视频会议(包括现有的WebRTC)中优于最先进的基于rl的方法的性能增益。我们的研究结果表明，R-FEC在视频速率和视频质量方面分别比最先进的解决方案高出27%和6dB。

2. Background

自2011年在Chrome浏览器中添加了WebRTC以来，大多数主流浏览器，如Firefox、Safari等都采用了它[31]。因此，WebRTC实际上是通过web浏览器进行视频会议的标准。在本节中，我们介绍了WebRTC中使用的拥塞控制算法，该算法确定了包括视频速率和丢失恢复速率(§2.1)在内的整体发送速率，并描述了WebRTC如何处理数据包丢失(§2.2)。

2.1 CC in WebRTC

WebRTC使用GCC算法[4,11]来确定目标速率。GCC利用丢包和延迟梯度来决定网络动态下的目标速率。特别是，GCC利用RTP(实时传输协议)/ RTCP (RTP控制协议)的延迟梯度进行队列延迟估计，这是推断拥塞的关键因素。GCC由两个控制器组成：基于丢包的控制器和基于延迟的控制器。

- 基于丢包的控制器：根据RTCP报告中丢失的RTP包数$f_l(t_k)$计算发送速率，变量为:

$$A_l\left(t_k\right)=\begin{cases}A_l\left(t_{k-1}\right)\left(1-0.5f_l\left(t_k\right)\right), & f_l\left(t_k\right)>0.1\\ 1.05\left(A_l\left(t_{k-1}\right)\right), & f_l\left(t_k\right)<0.02\\ A_l\left(t_{k-1}\right), & \text{otherwise}\end{cases}$$

其中t_k表示第k个反馈在发送端被接收到的时间。

（1）当丢包率很高，也就是$f_l(t_k)$大于0.1时，速率A_l乘性减少。

（2）当丢包率很低，也就是$f_l(t_k)$小于0.02时，速率A_l乘性增加。

（3）当丢包率中等时，速率不变。

- 基于延迟的控制器：它由三个部分组成：到达时间滤波器、过度使用检测器和速率控制器。到达时间滤波器跟踪单向延迟变化来估计排队延迟的趋势。overuse检测器根据时延梯度阈值到达时间滤波器的估计，将网络状态分为:过度使用、使用不足和正常。最后，速率控制器根据过度使用检测器的判定计算出发送速率A_d，公式如下：

$$A_d(t_i) = \begin{cases} \eta A_d(t_{i-1}), & \text{Increase} \\ \alpha R(t_i), & \text{Decrease} \\ A_d(t_{i-1}), & \text{Hold} \end{cases}$$

其中t_i是第i个帧在接收端被接收到的时间，其中$\eta = 1.08, \alpha = 0.85$。$R(t_i)$是过去500ms中的接收速率。

通过这两种控制器，GCC选择了一种保守的策略，也就是两者的最小值$A = min(A_l, A_d)$，其中A是视频编码的最终目标。

通过这种方式，GCC有望同时为视频会议提供高视频速率和低延迟。然而，根据WebRTC中的GCC性能调查[30]，即使在良好的网络条件下，由于基于延迟的控制器受到传输延迟的微小波动的影响，或者基于损耗的控制器受到较小的损耗的影响，发送视频速率也经常出现不必要的下降。

2.2 Handling Packet Loss in WebRTC

视频会议中的QoE容易受到丢包和延迟的影响，因为已知超过400ms的单向端到端延迟会损害用户之间的实时交互[14]。这就是为什么在视频会议应用中设计一个有效的损失恢复机制是至关重要的。为了满足严格的延迟要求，WebRTC采用了NACK (Negative acknowledgement) / FEC (Forward Error Correction)混合方案[12]。在混合NACK/FEC方案中，当RTT小于帧速率的倒数时，NACK比FEC更受青睐。因此，一种混合的NACK/FEC方法使WebRTC能够平衡NACK的延迟成本和FEC的冗余成本;在低RTT网络中，数据包可以重传而没有实质性的延迟损失，从而减少FEC数据包的数量。

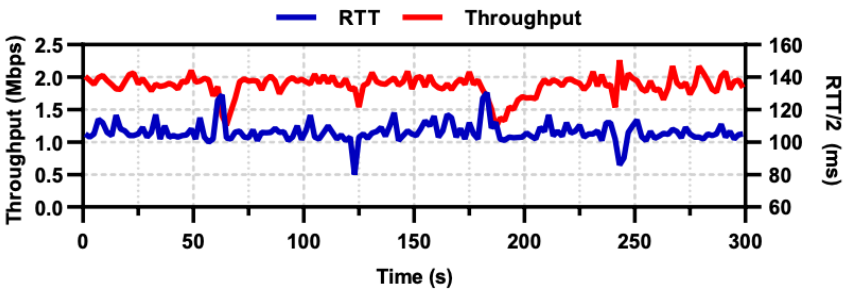
WebRTC使用媒体优化(Media Optimization, MO)模块控制自适应混合NACK/FEC方法。更具体地说，MO接受来自GCC的目标比特率作为输入，并接收来自RTCP数据包的网络反馈，例如数据包丢失的一小部分和往返时间(RTT)。根据接收到的统计数据和预定义的FEC/视频比表，MO设置FEC数据包的数量，并根据估计的开销更新编码器。然后，将FEC速率从估计的发送速率中减去，并将新的编码目标比特率设置为这个新的估计。请注意，WebRTC通过将开销限制在发送总比特率的50%来强制FEC数据包的数量不超过视频速率。最后，FEC编码器使用异或操作。FEC速率是通过调整组内的报文数来确定的，从而应用该操作生成FEC报文。

3. Key Observation

在本节中，我们通过实验验证了WebRTC的性能问题，主要关注拥塞控制和FEC速率控制。

3.1 Video Quality Degradation

如§2.1所述，我们观察到，即使在低RTT和损失率的良好网络条件下，实时WebRTC会话也不能保持高吞吐量[30]。我们使用我们的实验室网络测试台(详见§6.1)重现了实验以确认这个问题。为了保证良好的网络环境，我们通过100Mbps的有线链路连接两个最终用户，以100ms的传播延迟和无损耗为基本条件。当在两个用户之间运行WebRTC时，我们使用流量控制(TC)命令将传播延迟更改为小于30ms。具体来说，我们配置了四个延迟更改，每个更改在时间t=60、120、180和240时都小于30ms。



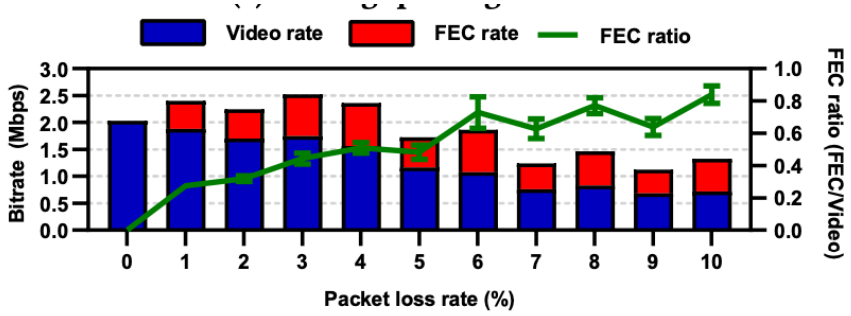
(a) Throughput degradation

在本实验中，WebRTC会话报告的平均传播延迟为105.88ms，没有丢包。图2a详细显示了实验中观察到的吞吐量和延迟。在所有四个故意的主要延迟改变点上观察到吞吐量下降。但是，吞吐量的减少在延迟增加然后减少的点比延迟减少然后返回的点更为明显。当延迟在t=60s和180s急剧增加时，吞吐量下降约50%。这是因为GCC错误地判断了由于延迟变化导致的网络拥塞。因此，GCC立即降低比特率以避免网络拥塞。这种突然的比特率下降会导致视频质量下降，最终导致用户的QoE下降。

3.2 Excessive FEC Overhead

通过采用混合NACK/FEC方法，WebRTC有望在视频质量和应用级延迟之间实现更好的权衡。如果重传仍然满足视频会议的严格延迟要求，NACK尤其有效，研究表明，对于RTT/2 < ~ 50ms的网络，NACK显著降低了FEC开销[12]。FEC主要用于RTT/2 > 50ms的网络，重传会导致无法忍受的延迟。根据基于测量的互联网延迟分析[32]，测量数据的延迟分布有三个RTT峰值，大约在45ms、135ms和295ms。考虑到视频会议连接全球用户的性质，大多数用户都受益于丢包情况下的FEC。为了研究WebRTC中的FEC开销，我们设置了一个100Mbps的有线链路，传输延迟为100ms。然后，我们分别测量视频和FEC数据包的比特率，同时在0% ~ 10%的丢包率范围内执行WebRTC。注意，当丢包率超过10%时，GCC会通过丢包控制器降低其发送速率。

图2b说明了WebRTC在丢包率小于10%的网络中的FEC开销。丢包率仅为4%时，FEC码率约为视频码率的50%。在丢包率为10%的情况下，FEC比特率达到视频比特率的80%左右;几乎每个视频包都是重复的，考虑到它大约每10个包丢失1个包，这远远超出了理想的范围。此外，FEC开销会影响接收端的延迟成本和发送端的视频比特率。这是因为WebRTC的MO模块控制由视频比特率和FEC比特率组成的总发送速率。结果，我们观察到视频比特率降低，而FEC开销增加。因此，找到一个最佳FEC率是必要的。



(b) FEC overhead

Figure 2: WebRTC performance investigation.

4. System Design

我们设计了一种RL算法来生成视频编码的目标比特率，并确定丢包情况下视频包恢复的FEC比率。本节首先介绍了我们系统的架构，然后介绍了系统设计的挑战和注意事项，然后描述了我们基于强化学习的控制算法。

4.1 System Architecture

图3显示了我们在替换现有GCC算法和FEC函数的同时运行在WebRTC之上的系统。来自网络摄像头的原始帧以初始目标比特率编码，并打包成RTP数据包(步骤1)。随后，RTP报文从发送方发送到接收方接收。然后，这些RTP包在接收缓冲区中进行处理，以检查包的顺序和时间戳、帧边界等。一旦帧重建完成，它被发送到解码器显示在用户的屏幕上。这里，接收器缓冲区负责RTCP反馈(步骤2)。到目前为止，我们使用的是WebRTC平台，所以工作流程与WebRTC相同。然后我们修改工作流，采用我们的RL代理来决定编码目标比特率和FEC比率。基于RTCP反馈和编码器的编码统计，我们构建了状态s_t，并将其馈送到RL代理(步骤3)。根据状态-动作映射策略(s_t, a_t)，我们的RL代理决定采取目标比特率和FEC比率的组合动作a_t(步骤4)。之后，使用RL代理的决策重复该工作流。

4.2 Design Considerations

我们提高接收端QoE的最终目标是通过使RL算法选择合适的视频比特率并确定不同网络条件(即带宽和数据包随时间的丢失$t \in T$)的最佳FEC比特率来完成的。我们将QoE最大化问题定义如下:

$$(\mathrm{P0}):\ \max_{\pi} \sum_{t=1}^{T} \{\alpha v_t - (1-\alpha) d_t\}$$

其中v_t和d_t分别为t时刻的归一化视频比特率和归一化延迟梯度。$\alpha \in (0,1)$是权重的超参数(但固定为0.5)。为了最大限度地提高接收端的总比特率，我们试图同时最小化超过t的延迟梯度，因为当不正确添加FEC数据包时延迟会增加。图4显示了FEC在不同损失率下对总延迟的影响。我们发现FEC数据包的数量会影响处理延迟。因此，我们还利用延迟梯度(最初用于估计WebRTC中带宽的过度使用或使用不足)来确定平衡的FEC使用情况。然而，由于以下几个实际挑战，找到一个最优策略(即所有t的视频和FEC速率)来解决这个目标是非平凡的:

- 挑战#1:由于反向链路类型、容量、延迟等原因，真实网络是高度动态的。生成具有几个固定阈值的规则集来确定统一适合所有网络情况的适当比特率几乎是不可行的。
- 挑战#2:FEC用于从丢包中恢复视频数据包。然而，FEC也引入了开销，因为FEC的冗余信息增加了所使用的带宽，并可能导致网络拥塞下的丢包增加。此外，过度使用的FEC数据包降低了有效的视频速率，并且由于其处理开销而增加了总体延迟。因此，找到FEC的平衡使用是具有挑战性的。
- 挑战#3:由于用户的实时交互，视频会议应用需要非常低的视频流延迟。由RL系统引起的时延开销(例如，通过RL算法进行决策)不应干扰用户之间的交互。
- 挑战#4:选定的视频和联邦利率不仅影响当前的QoE，还影响未来的政策选择。换句话说，需要一个能够改善总QoE的面向未来的解决方案，而不是根据当前状态制定策略来解决P0。

为了解决上述挑战，我们将问题转换为马尔可夫决策过程(MDP)，其目标是获得使累积奖励最大化的策略。由于MDP中假定存在的转移概率模型实际上是未知的，因此我们使用异步优势Actor-Critic (A3C)[19]算法，这是一种使用深度神经网络(DNN)的无模型RL算法。

4.3 State, Action and Reward Design

为了将原问题P0转化为MDP，我们定义状态、动作和奖励如下：

State。t时刻的输入状态被定义为$s_t = \left(\vec{v}_t, \vec{f}_t, \vec{d}_t, \vec{l}_t\right)$，它们分别代表视频比特率、帧率、延迟、丢包率。比特率和帧率可以直接从编码器得到，延迟和丢包率则是从RTCP的反馈中得到。对于每个状态，我们选用历史长度为5，也就是说过去的四个时刻被包括进来。使用历史在MDP建模中很流行，因为历史允许每个状态符合马尔可夫属性，从而提高对下一个状态的预测能力。实际上，历史上超过5的大小并没有显示出明显的好处。我们将所有特征连接到一个一维数组中，然后将它们提供给Actor和Critic网络。

Action。在s_t的基础上，agent做出行动a_t，表示了目标编码比特率和FEC比率的结合。我们把目标比特率的范围设置为100-2500kbps、7组，FEC ratio设置为0到80%、5组。因此，Action Space的大小为35，其中的一个组合被作为最终的输出。需要注意的是，FEC最大比率被限制到80%，这是根据图2（b）的观察得到的。

Reward。求解P0可以得到：

$$r_t = \alpha \sum_{n=1}^{N} v_n - (1-\alpha) \sum_{n=1}^{N} d_n$$

在Eq. 5中，N为当前时隙和N-1个过去时隙的观测次数，设为5，与上面的历史大小相同。为此，所提出的MDP的目标是找到一个使$\sum_{t=1}^{T} \{r_t\}$最大化的最优策略π(行动的时间序列)。

4.4 RL-based Video and FEC Rate Control

由于算法的性能受到神经网络结构的影响，我们在不同数量的神经元和隐藏层的组合中寻找性能最好的神经网络参数。我们为每个神经元和层的数量考虑了3种不同的架构。表1显示了一个简单的合成网络的平均奖励，它逐渐减少容量，然后又增加容量。有趣的是，有256个神经元的1个隐藏层表现出最好的性能，所以我们将这个配置设置为默认的神经网络架构。