清 华 大 学

综 合 论 文 训 练

题目：智能网络系统的实用性与可解释性研究

系　　别：电子工程系

专　　业：电子信息科学与技术

姓　　名：孟子立

指导教师：毕　军 教授

2019 年 6 月 2 日

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：学校有权保留学位论文的复印件，允许该论文被查阅和借阅；学校可以公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存该论文。

(涉密的学位论文在解密后应遵守此规定)

签　名：孟子正　导师签名：王[illegible]　日　期：2019.5.22

中文摘要

基于深度强化学习的智能网络系统在不同场景已经展现出很大的潜力。然而，本文作者发现，对于网络系统来说，深度强化学习中的深度神经网络仍存在过于复杂以及不透明等问题。其长决策延时、高资源消耗、以及较差的管理灵活性使得基于深度强化学习的智能网络系统难以在实际中大规模部署。在本文中，作者提出 TranSys 方法，通过高保真地将基于深度强化学习的智能网络系统中的深度神经网络转换为轻量级、透明化的模型以便于实际部署。TranSys 的关键思路是在训练时仍旧离线训练深度神经网络，但在线上部署时，将转化过的、可以精确模仿深度神经网络行为的模型进行部署。在两个有代表性的基于深度强化学习的智能网络系统上的实验表明，与传统基于深度神经网络的模型相比，基于 TranSys 的智能网络系统可以同时实现高保真度、短决策延时、低资源消耗、高透明度，却只引入可忽略的额外代价。例如，将 TranSys 应用于最先进的基于深度强化学习的自适应比特率调整视频客户端，与原客户端相比，可以减少额外的页面加载时间至原来的 156 分之一、动态内存消耗至 6.6 分之一同时将性能损失控制在 0.6%之内。

关键词：轻量级；透明可解释；基于深度神经网络的智能网络系统；决策树。

ABSTRACT

While Deep Reinforcement Learning (DRL)-based networked systems have shown a great potential in various application scenarios, a key observation is that Deep Neural Networks (DNNs) in DRL are heavyweight and nontransparent for networked systems. The long decision-making latency, high resource consumption, and poor manageability together prevent DRL-based networked systems from deploying in practice. In this paper, we propose TranSys, a method that can faithfully translate the DNNs in DRL-based networked systems into lightweight and transparent models for practical deployment. The key idea of TranSys is to train a DNN offline, but deploy the translated model online that can faithfully imitate the behaviors of the DNN. Evaluation results on two representative DRL-based networked systems demonstrate that TranSys-based networked systems could achieve high faithfulness, short decision-making latency, low resource consumption and high transparency compared to original DRL-based networked systems with negligible additional overhead. For example, applying TranSys over a state-of-the-art DRL-based adaptive bitrate video client reduces the additional page load time by 156× and runtime memory utilization by up to 6.6× while maintains the performance loss within 0.6% compared to the original DRL-based client.

Keywords: Lightweight; Transparent and Interpretable; DNN-based networked systems; Decision tree.

目 录

主要符号对照表

ABR	自适应比特率调整（Adaptive Bitrate）
API	应用程序编程接口（Application Programming Interface）
ASIC	特定应用的集成电路（Application Specific Integrated Circuit）
CART	分类和回归树（Classification and Regression Tree）
CNN	卷积神经网络（Convolutional Neural Network）
CPU	中央处理器（Central Processing Unit）
DAG	有向无环图（Directional Acyclic Graph）
DASH	基于 HTTP 的动态自适应流传输（Dynamic Adaptive Streaming over HTTP）
DNN	深度神经网络（Deep Neural Network）
DRL	深度强化学习（Deep Reinforcement Learning）
DSP	数字信号处理器（Digital Signal Processor）
FCT	流完成时间（Flow Completion Time）
GNN	图神经网络（Graph Neural Network）
GPU	图形处理器（Graphics Processing Unit）
HTTP	超文本传输协议（HyperText Transfer Protocol）
ILP	整数线性规划（Integer Linear Programming）
MLFQ	多级反馈队列（Multi-Level Feedback Queue）
NFV	网络功能虚拟化（Network Function Virtualization）
NIC	网卡（Network Interface Card）
QoE	用户体验（Quality of Experience）
RMSE	方均根误差（Root-Mean-Square Error）
RNN	循环神经网络（Recurrent Neural Network）
RTT	往返时间（Round-Trip Time）
SDN	软件定义网络（Software Defined Networking）
SLA	服务等级协议（Service Level Agreement）

第1章 引 言

1.1 研究背景

最近，有许多研究工作提出了将深度强化学习应用于网络系统中，来提升其性能。在网络系统的各个领域，研究者们通过将领域内的具体问题与深度神经网络（或深度强化学习）的特性相结合，在一些针对于领域的优化的基础上，提出了许多代表性的基于深度强化学习的智能网络系统。其中，取得了较为成功的应用的领域包括视频传输、流量优化以及任务调度：

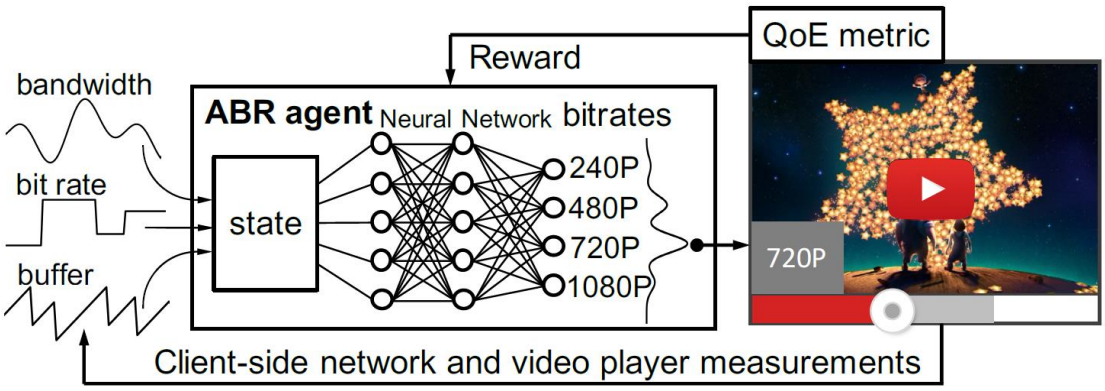


图 1.1 Pensieve[53]系统结构

在视频传输领域， Pensieve[53]通过将深度强化学习应用于自适应比特率调整算法，通过用神经网络替代原有决策，将用户体验在不同场景下提升了 12%-25%。awDNN[80]和 NAS[81]提出，可以通过超分辨率恢复等手段，在视频播放器中引入深度神经网络，来改善视频质量。

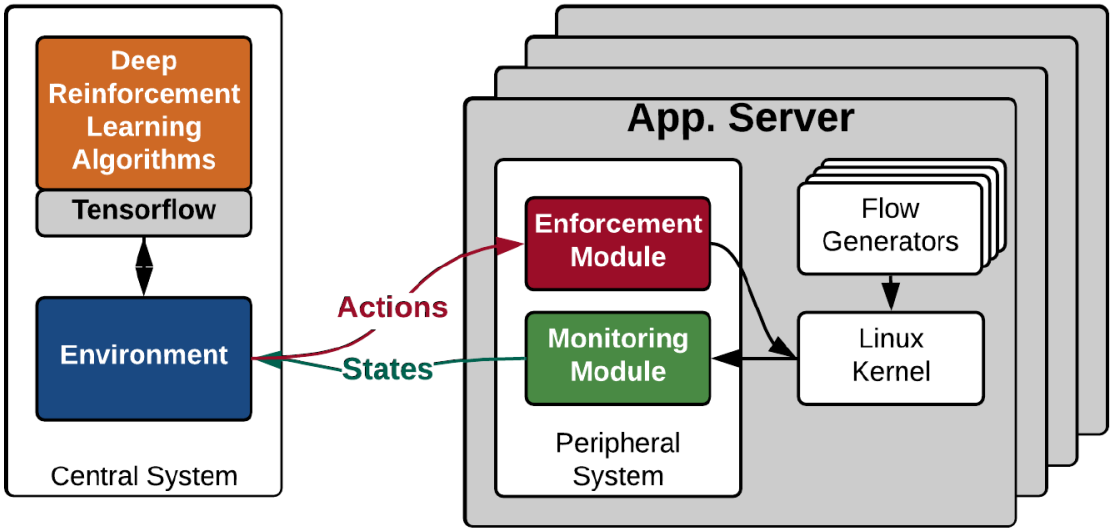


图 1.2 AuTO[21]系统结构

在流量工程领域，AuTO[21]通过在集中式的控制平面内采用深度强化学习的方法，来精确地对存在时间较长的网络流进行基于深度神经网络的逐流控制，并结合线上学习，使得系统能够较好地适应动态网络环境。其流完成时间与现有算法相比，最多可降低 48%。DeepConf[67]则利用深度强化学习，通过结合数据中心网络拓扑的可改变性，来动态配置网络拓扑。Softmin[76]则对网络流量调度时的流分割比例进行了优化。

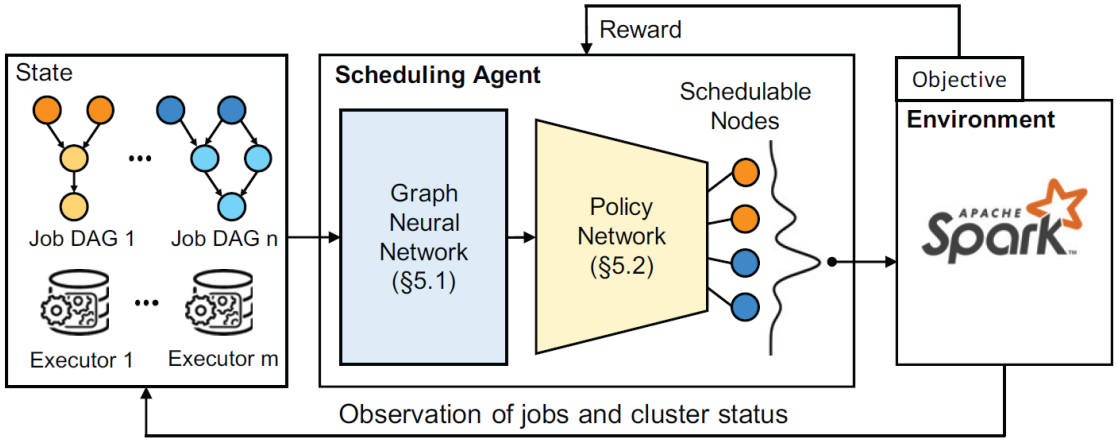


图 1.3 Decima[54]系统结构

在并行任务调度方面，最早由 DeepRM[52]提出可以采用深度强化学习对资源调度进行优化，提出了初步思路。Decima[54]则针对于任务调度时任务连续到达而非在最开始时刻全部知道的性质，并针对任务的有向无环图的表示进行优化，可将平均任务完成时间缩短最少 21%。DQ[44]则进一步针对查询系统的特性，利用深度强化学习进行了优化。

在深度神经网络的强大的预测能力的支持下[55][56]，基于深度强化学习的智能网络系统已经有了很多成功的应用。相比于现有的基于启发式算法的解决方案，这些应用在复杂网络优化中取得了很大的性能提升。

1.2 研究现状

然而，直接将深度强化学习应用到网络系统中存在如下两个问题[2.3 节]。一方面，对于现有网络应用来说，深度神经网络过于复杂。例如，将一个基于深度强化学习的智能比特率自适应调整系统[53]集成到视频播放器中会带来十余秒的额外页面加载延迟。用户可能会在等待中丧失耐心而离开当前页面[45]。类似地，部署一个基于深度强化学习的逐流调度系统[21]会带来大约 100 毫秒的决策延迟，而在这一过程中大于 95%的流已经完成，这导致事实上大部分逐流调度决策是没有用的。另一方面，深度神经网络高级复杂的黑盒子结构使得网络管理员很难去实际管理基于深度强化学习的智能网络系统。例如，当深度神经网络产生欠优的或者不正确的输出结果时，由于深度神经网络的决策过程是不透明的[84]，网络管理员很难去准确找到问题的原因。这两个问题使得深度强化学习在网络系统的实际部署还有很长的路要走。

尽管在网络领域，利用深度神经网络来提升网络性能仍在初级阶段，但在网络领域之外，如何将大规模深度神经网络部署到资源高度受限的设备上（如：物联网设备）受到了关注。学者提出的方法包括将决策过程卸载到远程云端[38][53]、引入新的硬件加速设备[5][24][46]，和通过去除不显著的神经元来压缩神经网络[18][23][31][47]等等。然而，正如作者在 2.5.1 节中讨论的那样，如果在网络领域采用上述解决办法，在云端远程进行决策会增加服务器资源和网络流量传输的成本，采用特定的加速设备既昂贵又不可扩展。压缩神经网络则可能会带来潜在的性能下降[31][47]。同时，他们也并未考虑到网络系统的特性（如：序贯决策过程），效果不佳。

1.3 研究思路与贡献

受可解释性机器学习方法的研究进展启发[22][37][63][83]，我们发现上述挑战可以通过将深度神经网络转换为轻量级而又透明的模型（如局部回归[37][63]、决策树[12]）来解决。这一转换的理论基础是：有些模型（例如：决策树、一些符号语言等）尽管其表达形式更为简单，但其表达能力和深度神经网络相差无几[10]。这一表达能力的相似性使得我们可以高保真地将已经训练并精调过的深度神经网络在具有最坏情况下性能的理论保障的情况下[13]转换为轻量级且透明的模型。这组成了 TranSys 的核心思路：一方面我们在训练时仍然使用深度神经网络来保证性能，而另一方面我们将转换后的模型进行线上部署。在这种情况下，长决策延时和高资源消耗的问题就可以被解决了。同时，转换后的模型的透明性使得网络管理员可以高效地管理基于深度强化学习的智能网络系统。然而，尽管可解释性方法可以为深度神经网络提供具有透明性的近似模型，由于以下原因，将基于深度强化学习的智能网络系统转换仍然是新颖且具有挑战的：

- **不适用于深度强化学习。**大部分现有解释性方法是为监督学习而设计[6][63][83]。然而，他们很难被直接应用到深度强化学习中：监督学习是优化单个决策的损失函数，而强化学习是优化一个策略。
- **缺乏表达能力。**网络系统的策略的内部逻辑通常比诸如线性拟合等现有解释性方法[37][63]所能表达的要复杂[3.1 节]。直接将基于深度强化学习的网络系统的策略转换为上述方法会导致性能上的损失[4.2 节]。
- **对连续输出不适用。**与现有解释性方法中所分析的离散分类结果所不同，有时网络系统的决策是连续变量（如：队列阈值[21]、流分割比例[76]），这会让问题更加具有挑战性。

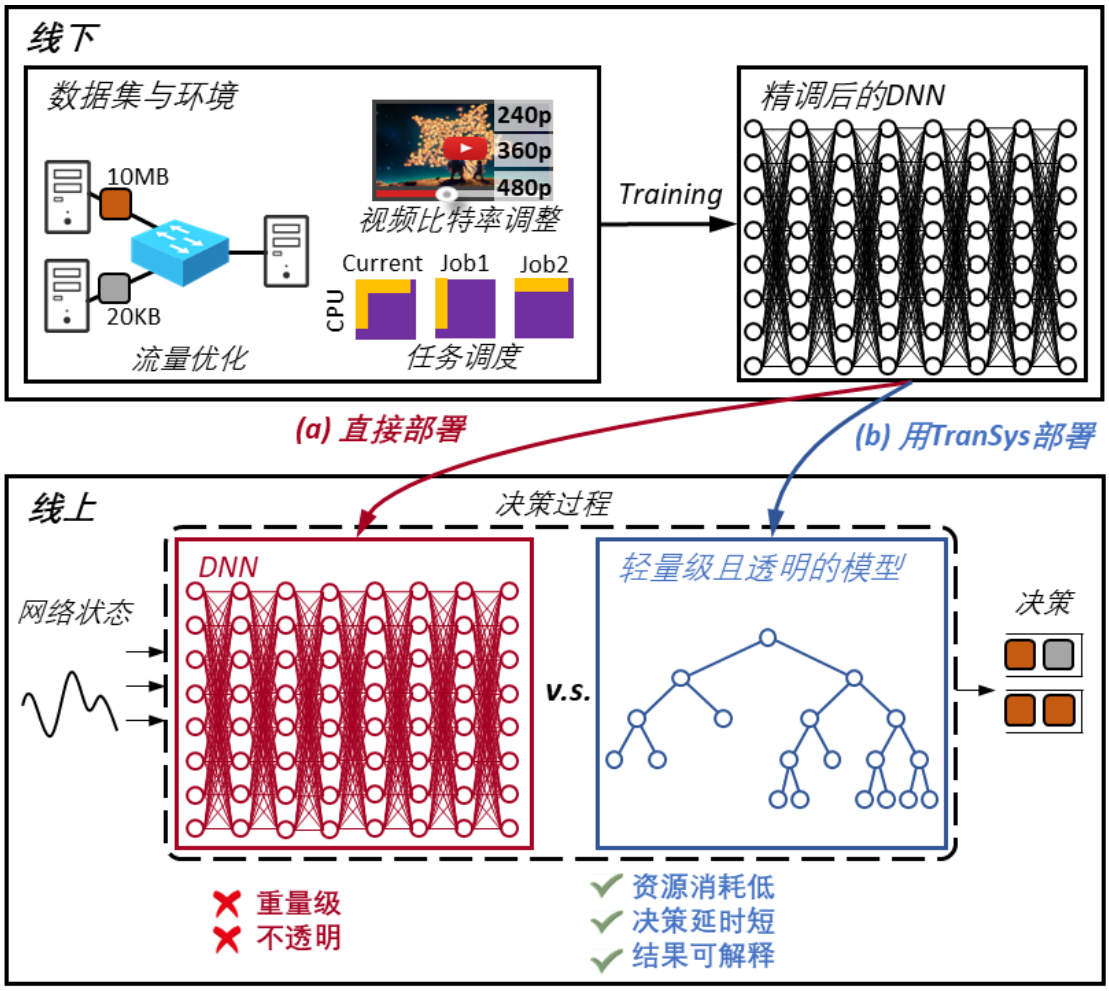


图 1.4　现有基于强化学习的智能网络系统的部署过程(a)和 TranSys(b)

作为回应，我们提出 TranSys，一种普适性的框架来高保真地将基于深度强化学习的智能网络系统转换为轻量级和透明的系统。如图 1.4 所示，在 TranSys 中，网络管理员在最开始的时候会和往常一样来离线训练一个深度强化学习模型，然后将深度神经网络转换为一个轻量级且透明的模型（图 1.4 中的分支(b)）。转换后的模型会高保真地模仿深度神经网络的决策，同时也有轻量级和透明化的优点。在本文中，作者采用决策树来作为转换后的模型。采用决策树背后的动机是决策树的结构和网络系统中的决策过程很类似[3.1 节]。这使得决策树可以高保真地模仿深度神经网络的决策，并在实际部署中保持和原有基于深度强化学习模型类似的性能。更进一步地，为了解决支持连续输出的问题，我们引入回归树来在 TranSys 中预测连续实数变量。

除了高保真地转换之外，TranSys 也可被用来提升基于深度强化学习的智能网络系统。根据本文的实验结果，我们简要总结了 TranSys 在当前基于深度强化学习的网络系统中的三点提升。第一，我们将 TranSys 应用于一个基于深度强化学习的流调度系统[21]。和原系统相比，其将决策延迟缩短至 27 分之一、将流完成时间缩短了至多 8%[4.3 节]。第二，我们将一个基于深度强化学习的自适应比特率调整系统[53]中的深度神经网络转换为决策树，这将额外的页面加载时间缩短至原来的 156 分之一，同时保持了和原来基于深度强化学习的智能网络系统几乎相近的性能[4.4 节]。第三，我们利用 TranSys，对一个基于深度强化学习的自适应比特率调整系统进行故障检测，并仅用决策树将平均用户质量提升了 3%[4.5.1 节]。

简而言之，本文做出了以下三点贡献：

- 我们总结了现有基于深度强化学习的智能网络系统的两个关键问题：在实际部署中，他们过于重量级且不透明。对于深度神经网络来说，这意味着高资源消耗、长决策延时、弱管理能力，使得其很难在实际中部署[第 2 章]。
- 我们提出了 TranSys，一种通用性的方法，通过将深度神经网络转换为轻量级和透明的模型，来让基于深度强化学习的智能网络系统更加轻量级和透明化。在基于 TranSys 的网络模型中，我们在线下仍旧训练深度神经网络，但是在线上部署转换后的模型[第 3 章]。
- 我们在两个代表性的基于深度强化学习的智能网络系统上面部署了 TranSys。我们的实验结果表明，与原有系统相比，基于 TranSys 的智能网络系统可以同时达到高保真度、短决策延迟、低资源消耗、高透明度[第 4 章]。

1.4 论文内容

为验证上述研究思路，本文设计了一套对基于深度强化学习的智能网络系统的转换方法——TranSys 系统。整篇论文将对此系统的设计与实现进行详细介绍，本文具体结构如下：

第一章为引言，主要介绍本研究的背景、研究现状以及 TranSys 的研究思路于方案。

第二章讨论 TranSys 系统设计背景以及设计的具体动机、及要解决的现有网络系统中存在的问题。同时归纳总结了与本设计相关的几类工作，并讨论了本工作与相关工作相比有何创新之处。

第三章描述了 TranSys 系统在转换深度神经网络到其他模型中的设计选择以及采用的设计方案。

第四章介绍了如何将 TranSys 部署到现有基于深度强化学习的智能网络系统上。我们还通过多组实验来证明了 TranSys 在保真度、决策延迟、资源利用、透明性以及部署代价上均具有较好的效果。

第五章简要讨论了 TranSys 的可扩展性于应用场景。

第六章总结了整篇论文的工作，并展望了未来工作。

尽本文作者所知，TranSys 是第一个通用地实用化、解释并提升基于深度强化学习的智能网络系统的方法。我们在将深度神经网络在计算机网络的应用中展开了一些初步的讨论。我们相信 TranSys 会加速基于深度强化学习的智能网络系统的实际部署。在未来我们将开源 TranSys，并希望 TranSys 可以在这一领域激发出在不同应用场景下的后续工作。

第2章 相关研究综述

2.1 本章引言

在本章中，我们将首先介绍深度强化学习的技术背景与优势[2.2 节]。我们接下来介绍几个代表性的基于深度强化学习的智能网络系统[2.3 节]。接下来，我们总结了当前基于深度强化学习的智能网络系统的两大问题，来推动 TranSys 的设计[2.4 节]。最后，我们调研了这两大问题目前的相关文献综述，并对比介绍了本文的贡献与创新点[2.5 节, 2.6 节]。

2.2 深度强化学习背景介绍

我们通过一个基于深度强化学习的网络流调度器[21]来解释深度强化学习。在第t次迭代中，智能体（流调度器）首先从周围环境中观察到状态s_t（活动流的五元组和优先级）。智能体接下来会根据其策略π做出动作决策a_t（对当前周期内的活动流来说，就是优先级和速率限制）。环境紧接着会返回奖励r_t（当前周期内的平均流完成时间）并更新其状态到s_{t+1}。奖励用来指示当前决策的好坏。优化的目标是学习一个策略π，使得累计未来折减奖励$\mathbb{E}[\sum_t \gamma^t r_t]$ $(\gamma \in (0,1])$最大。$\pi_\theta(s,a)$是在π_θ策略下，当观察到状态s时，做出决策a的概率。其中，θ为参数，在解决大规模实际问题中一般用深度神经网络来表示[55][56]。关于深度强化学习的更多背景知识，读者可以参考[73]。

和监督学习相比，在将机器学习应用到网络系统中时，深度强化学习吸引了更多研究工作的注意。我们发现，深度强化学习的广泛采用的背后有如下两个原因：

- **序贯决策过程。**深度强化学习研究的是一个智能体在给定环境下，如何去通过不断收集当前状态，并做出一系列决策，以达到相应目标的过程。过去的成功应用案例（如：Atari[55]和 AlphaGo[56]）都是序贯决策过程。同时，网络系统也有着相似的过程：连续地收集网络测量数据，以优化流完成时间或用户体验（Quality of Experience, 简称 QoE）为目标，在流调

度、视频比特率选择等方面做出一系列选择。网络系统的实质和深度强化学习的应用场景之间存在很高的相似性。

- **无明确及时反馈。**与图像分类或自然语言处理不同，网络系统的评估一般在策略层面而非单个决策层面。例如，在不考虑未来的网络状况的情况下，我们很难判断一个独立的比特率选择决策是好是坏。因此，和监督学习的应用场景不同（如：图像分类[28]、恶意软件检测[37]），对于单个决策而言，在网络系统中并没有评价标准。网络系统关注的是一个策略的好坏。这和深度强化学习的应用场景吻合：深度强化学习的优化目标恰恰是累计折减未来奖励$\mathbb{E}[\sum_t \gamma^t r_t]$，与网络系统的优化目标一致。

2.3 基于深度强化学习的智能网络系统

表 2.1 一些基于深度强化学习的智能网络系统

应用场景	例子
视频传输	Pensieve[53], NAS[81], awDNN[80]
流量工程	AuTO[21], Softmin[76], DeepConf[67]
任务调度	DeepRM[52], Decima[54], DQ[44]

受深度强化学习模型的高性能所激励，最近的许多研究工作都将深度强化学习应用在网络领域的不同场景中。我们将其中的一些代表性工作总结如表 2.1 所示。上述工作在相关领域内对状态空间和决策空间进行了建模，并针对性地采用深度强化学习算法来分别获得其优化目标。为加速这一训练过程，网络管理员通常需要开发一个仿真器[34][66]来对真实环境进行仿真[44][53][54]。在本文中，为了证明 TranSys 的普适性，我们聚焦于近年来开发的两个较具有代表性的基于深度强化学习的智能网络系统：

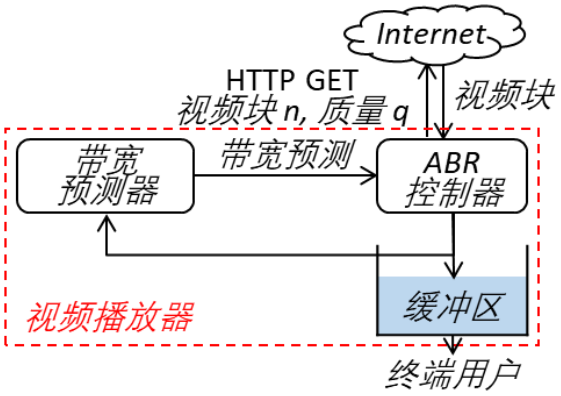


图 2.1 DASH 机制[72]。

Pensieve[53]。现阶段互联网视频传输机制被标准化为基于 HTTP 动态自适应的流传输（Dynamic Adaptive Streaming over HTTP，简称 DASH）[72]。在这之中，每个视频包含若干小块（几秒钟的播放时间），每个小块都被编码为若干个比特率。如图 2.1 所示，客户端的视频播放器采用自适应比特率调整算法，来根据带宽变化情况优化用户体验[53][81]。Pensieve 是一个基于深度强化学习的自适应比特率调整系统，其基于网络测量的结果，如上一块平均带宽、缓冲区占用情况，来对用户体验进行优化。Pensieve 比现有工作可以提升至少 12%的用户体验。

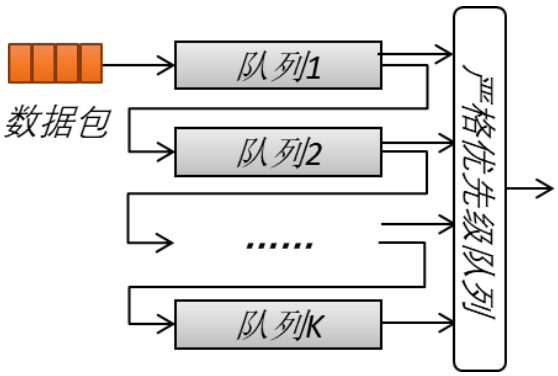


图 2.2 多级反馈队列[11]。

AuTO[21]。AuTO 是一个采用深度强化学习来优化流完成时间的可扩展的流调度系统。AuTO 采用了多级反馈队列（Multiple Level Feedback Queue，简称 MLFQ）的设计[11]。如图 2.2 所示，在多级反馈队列中，一个新流从最高优先级开始，当其发送字节数超过一定阈值后，就会下降到较低级别的队列中。受深度强化学习的长决策延迟所限，AuTO 只能通过长流强化学习智能体（Long-flow RL Agent，

简称 lRLA）来对长流进行逐流优化。对于短流，AuTO 仍采用多级反馈队列的形式进行处理，但是引入了短流强化学习智能体（Short-flow RL Agent，简称 sRLA）来对多级队列的阈值进行优化。lRLA 以活动流的{5 元组、优先级}以及完成流的{5 元组、流完成时间、流大小}作为输入，并为每一个活动流决定其{优先级、速率限制、路由路径}。sRLA 则以完成短流的{5 元组、流完成时间、流大小}作为输入，并输出多级反馈队列的阈值。在这种两层设计的帮助下，AuTO 将流完成时间在不同场景下缩短了最高 48.14%。

2.4 现有系统问题分析

尽管在基于深度强化学习的智能网络系统中采用深度神经网络可以提高网络系统性能，这也带来了四个系统层面的问题。

问题#1：决策延迟长。众所周知，相比于现有的启发式方法，深度神经网络对资源和运行时间的占用都极高[75]。然而，与深度学习应用的其他领域不同，由于数据包在网络中以极高的速度传输，网络系统通常对延迟有着苛刻的要求。在一条 10Gbps 链路上，1 毫秒的延迟会产生 1.25MB 的数据包堆积。例如，AuTO 的逐流调度决策延迟约为 100 毫秒。在这一过程中，数据中心中大部分数据流（>95%）均已经完成[4.3 节]。因此逐流调度系统很难被应用到大部分流中：对他们而言，逐流决策是无用的，这是因为决策延迟比他们的流完成时间还要长。AuTO 为解决这一问题，只好放弃针对短流的逐流调度决策，将控制范围仅缩小到那些长流。这会牺牲深度学习所带来的性能增益[4.3.1 节]。

问题#2：资源消耗高。在视频客户端上初始化 Pensieve 会带来大约 10 秒中的额外的页面加载时间[4.4 节]。这会使得一部分用户离开页面，导致内容提供商的收入下降[29][45]。同时，在视频客户端上部署 Pensieve，和现有部署方案相比，仅仅对于自适应比特率调整算法而言，就会带来接近 3MB 的额外内存消耗[4.4 节]。这会拖慢用户设备，并会消耗额外的资源和能耗[19]。为解决这一问题，Pensieve 引入了远程服务器来处理深度神经网络推断，因之在设备、传输流量等方面引入了高昂的开销[第 6 章]。同时，对于那些编程能力极度受限的网络设备（如：可编程交换机，可编程网卡）而言，深度神经网络很难在不进行重新设计的情况下直接部署。

问题#3：管理能力差。有与网络系统本身的复杂性，其故障检测问题已经成为困扰网络管理员与网络研究者数十年的问题。而由于深度神经网络的黑盒子的

特性[37][63]，将其引入网络系统只会让这一问题进一步恶化。深度神经网络经常包含成千上万个神经元[21][37]，由于其高复杂度，想要了解深度神经网络是如何做出一个决策是十分困难的。这限制了网络管理者对于基于深度强化学习的智能网络系统的管理能力：当深度强化学习模型做出一些有问题的决策时，网络管理者很难对故障进行准确定位[84]。这一不透明的决策过程同样使得基于深度强化学习的网络系统难以被完全信任，从而难以在实际中大规模部署。

问题#4：可解释性低。对于网络系统而言，通常来说现有采用的方案都是一些网络管理员可以直接理解的启发式方法（如：先到先服务调度），并取得了相对尚可的性能，在数十年的更新迭代中取得了很好的发展。然而对于基于深度强化学习的网络系统而言完全不同：网络研究者们甚至不知道是什么策略超过了现有算法。如果能够将深度神经网络的策略以网络管理员能够理解的方法解释出来，我们可以从中学习到设计新算法的启发。这一可解释性同样也会增强领域内专家部署智能网络系统的信心。

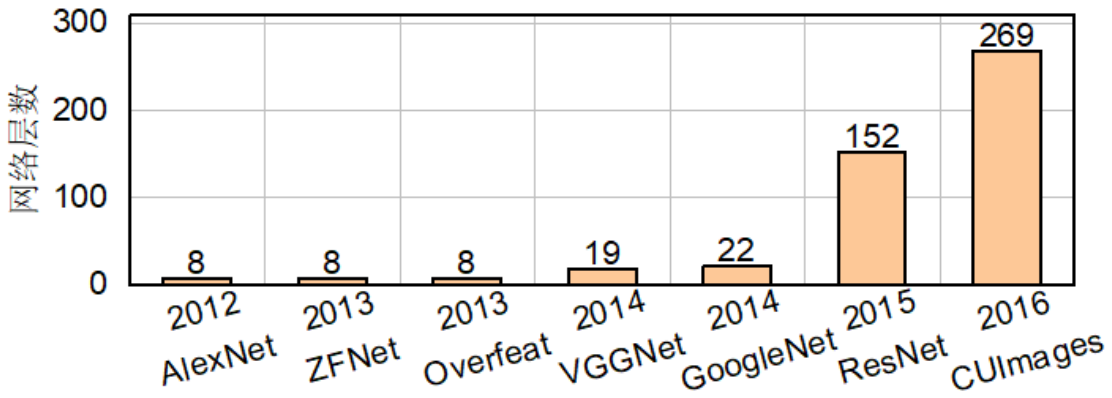


图 2.3　深度革命：近年 ImageNet 大规模视觉识别挑战赛冠军网络层数增长剧烈[28][30][39]。

总之，尽管基于深度强化学习的智能网络系统有较好的性能上的提升，在没有解决以上问题之前，在实际中部署基于深度强化学习的系统是很困难的。需要注意的是，目前基于深度强化学习的智能网络系统中的深度神经网络其实还处在应用的初级阶段：设计于 2017 年的 Pensieve 系统只有一个隐藏层，设计于 2018 年的 AuTO 系统也只有不超过 10 个隐藏层。作为对比，为了让神经网络更容易训练，在其他领域近年来神经网络层数迅猛增加，层数甚至可以达到成百上千层，如图 2.3 所示。在此处本文作者并不想试图强调越深的网络会带来越好的效果，但不可争辩的是，更深的网络会让这一问题进一步恶化。这驱动着我们要去将深

度神经网络转换成轻量级的、透明的模型，将基于深度强化学习的智能网络系统实用化。

我们将上述四个问题分为两类。决策延迟和资源消耗的问题可以通过将智能网络系统轻量化解决，而管理能力和可解释性的问题可以通过将智能网络系统透明化来解决。接下来，我们将围绕这两点问题进行研究。

2.5 深度神经网络预测时线上消耗

本节介绍了在解决深度神经网络的执行负担重方面，将深度神经网络轻量化的相关工作。已有工作提出，可以通过各种方法要么将线上消耗转移、要么将线上消耗隐藏。同时，本节对比已有工作，提出本文在深度神经网络轻量化方面的创新性。

2.5.1 轻量化解决方法

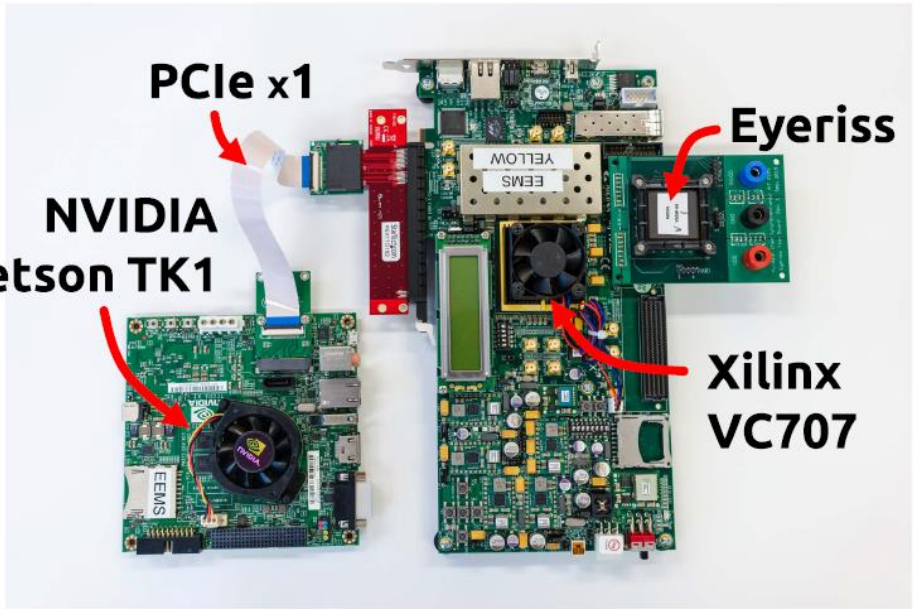


图 2.4 Eyeriss[24]：一种可重配置的神经网络加速器结构图

如前所述，如何将大规模深度神经网络部署到资源高度受限的设备上（如：物联网设备）在最近受到了前所未有的关注。学者提出的方法包括将决策过程卸载到远程云端[38][53]、引入新的硬件加速设备（如：DSP[46]、GPU[5]、ASIC[24]），和通过去除不显著的神经元来压缩神经网络[18][23][31][47]等等。然而，在云端远程进行

决策（如：[53]中的自适应比特率调整算法服务器）会增加服务器资源和网络流量传输的成本。采用特定的加速设备既昂贵又不可扩展。压缩神经网络则可能会带来潜在的性能下降[31][47]。

2.5.2 本文创新

在不抛弃将现有神经网络部署到线上的思路下，即使通过神经网络剪枝压缩等手段大幅度降低了神经网络的运算代价，上述已有工作仍然很难被部署到如网卡或交换机等编程能力极度受限的网络设备上。例如，可编程交换机上对浮点数运算的支持都非常初步。更进一步地，他们中没有方法能够提供 TranSys 所提供的可管理性和透明性。相比之下，本文通过将神经网络转换成决策树，从根本上降低了预测时的线上消耗，同时也具备了一定的可管理性和透明性。

2.6 深度神经网络可解释性研究

本节介绍了在解决深度神经网络不透明方面，提升深度神经网络可解释性的相关工作。

2.6.1 深度神经网络解释方法

图 2.5 LEMNA[37]：为神经网络分类结果做出解释

在我们注意到网络领域的这一问题的同时，最近也有许多研究工作希望在不同的领域内建立透明的、可解释的系统。例如，在图像分析领域[14][69][83]，研究者通过分析图片中具体哪些位置对最终决策产生了影响，以解释神经网络做出的结果，如图 2.5 所示。在自然语言处理[22][64]、推荐系统[20][25]和网络安全[35][37]等其他

领域，研究者们也通过各种手段和方法，以增强具体领域内管理员对于神经网络的输出结果的信心。

2.6.2 本文创新

然而，上述方法主要基于监督学习方法（如：带标签分类、回归等），因而很难被直接部署到基于深度强化学习的智能网络系统中。更进一步，上述工作主要关注于增强原有深度神经网络的可解释性，却忽视了其过于重量级的一面。这对网络系统而言很关键。尽我们所知，TranSys 是第一次同时兼顾轻量级和透明性，因此可广泛地将基于深度强化学习的智能网络系统实用化、解释并提升的方法。这将会加速基于深度强化学习的智能网络系统的部署。

第3章 深度强化学习模型转换关键技术研究

作为在将网络系统轻量化与透明化、从而实用化的初步尝试，我们的方法在很大程度上基于近期在深度强化学习的形式化验证上的工作[13]。我们针对我们的两个需求以及网络系统的特性对上述工作进行了改进。

3.1 本章引言

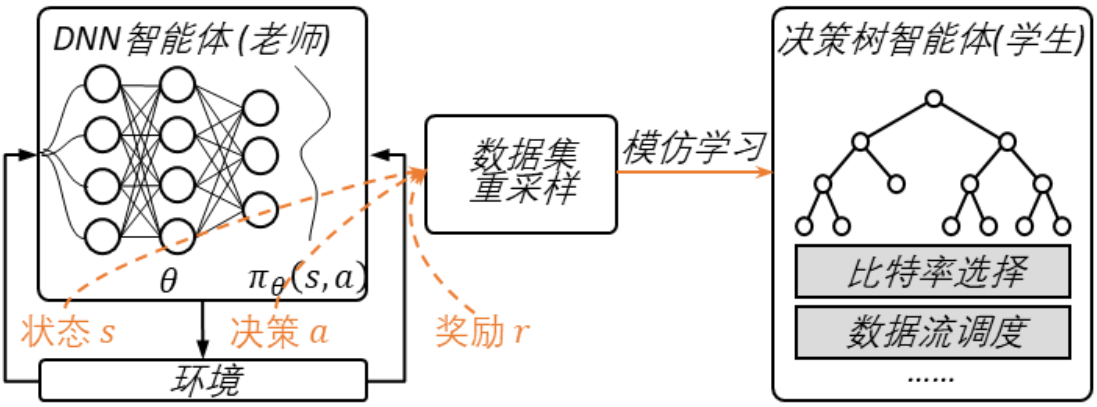


图 3.1 TranSys 转换设计

TranSys 转换方法的设计概览如图 3.1 所示。简而言之，TranSys 采用了模仿学习[41]，在一个已经训练好的深度神经网络智能体（老师）的指导下，产生一个决策树智能体（学生）。然而，如我们在第一章中所述，深度强化学习的损失函数和现有决策树算法所优化的损失函数[33][48]并不相同：深度强化学习优化的是当前策略的累计未来递减奖励，而决策树训练方法优化的是单个决策的准确度。为了直接、高效地在从深度强化学习模型收集到的数据集上采用当前决策树训练算法来指导学生的训练，我们首先需要对收集到的数据集进行重采样。在那之后，将学生在重采样过的数据集上进行训练，学生便可以在自己的结构上学到老师的决策策略。对于网络管理员来说，TranSys 包含以下步骤：

步骤1. 和往常一样，在仿真环境中或者在线上训练基于深度强化学习的智能网络系统[21][53]。

步骤2. 测试训练好的深度神经网络，并收集（状态、决策、奖励）三元组。

步骤3. 对三元组进行重采样以提高深度强化学习模型转换的保真度[13]。

步骤4. 利用模仿学习，将深度神经网络模型转换为一个轻量级且透明的模型[41]。

步骤5. 将转换后的模型进行部署，并同时获得深度强化学习带来的性能提升和转换模型带来的轻量级化与透明性。

在本章中，我们将首先介绍为什么我们选择决策树作为轻量级且透明的目标转换模型[3.2 节]。我们接下来介绍我们如何利用[13]中的方法，对深度强化学习收集到的元组进行重采样[3.3 节]，并产生一个决策树[3.4 节]。接下来，针对于网络系统而言，我们介绍如何在网络系统中利用回归树支持连续输出[3.5 节]。我们将部署和具体应用上的优化留到第 4 章进行介绍。

3.2 智能网络系统的设计选择——决策树

如第一章中所述，TranSys 基于可解释性工作，将深度神经网络转换为轻量级且透明的模型。可解释性方法包含为决策做出解释，和从头建立一个可以自我解释的且性能与深度神经网络相当模型[6]。与原有可解释性方法不同，TranSys 的一个观察是由于转换后模型一般较为简单，我们可以利用转换后模型的轻量级的特性。尽管轻量级这一特性也可以通过神经网络压缩的方法来进行达到[18][23][31][47]，他们并不能想可解释性方法一样，提供透明性这一对网络系统十分重要的特性[2.3 节]。根据他们是否需要模型结构的信息，可解释性方法可分为白盒和黑盒方法[37]。黑盒方法将模型（一般是分类器）看作一个黑盒子，并利用线性回归[63]、带权回归[49]、混合回归[37]等方式来分析模型的输入-输出关系。由于他们并不需要额外的信息，他们可以广泛地应用在任意的分类器上。相比之下，白盒方法则需要通过前向传播[35]或后向传播[70]等方法对神经网络模型分析。更多关于此的讨论，读者可参考[6]。

然而，根据我们的实验结果[4.2 节]，诸如回归的黑盒方法并不能为网络系统提供足够的表达能力，因此在基于深度强化学习的智能网络系统上保真度不佳。尽管白盒方法因为有更多的模型结构信息可以有较高的保真度，就现阶段而言，他们主要是为卷积神经网络[83]或循环神经网络[22]等监督学习方法所设计，对深度强化学习并不适合。为基于深度强化学习的智能网络系统选择一个高保真、透明的方法是有挑战的。

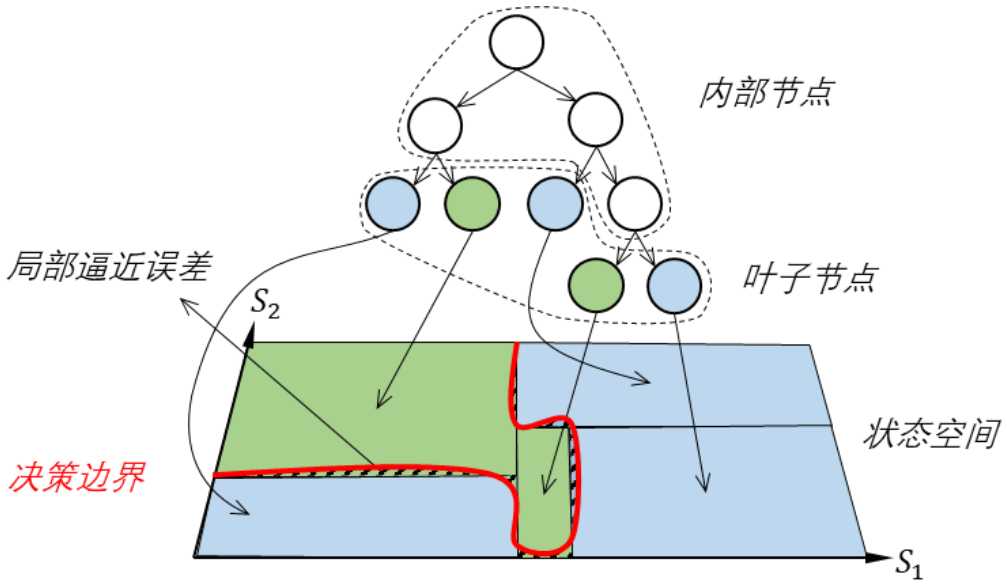


图 3.2　决策树可高保真地拟合非线性边界

在本文中，我们采用决策树作为转换深度神经网络的目标模型，是基于以下两个原因。其一，决策树的逻辑结构和网络系统的决策十分相似：他们通常都是若干判断条件的逻辑组合。例如，自适应比特率调整算法在当前缓冲区空闲情况、上次比特率选择结果和预测的带宽情况来综合做出判断[53][71][82]。其二，如图 3.2 所示，由于其非参数化的特性，决策树具有丰富的表达能力和较高的保真能力，因此可以表达非常复杂的策略[15]。我们通过在不同的基于深度强化学习的智能网络系统上将决策树和其他方法的实验比较来证明决策树转换深度神经网络的保真度[4.2 节]。关于决策树相对于深度神经网络的保真度和误差界的理论分析，我们推荐读者阅读[12][13]。同时，也有一些有潜力的方法通过符号回归[50]或者程序搜索[78]等方式来对深度强化学习进行解释。然而，他们一般需要领域内的专家知识集成进去，需要为每个应用单独设计，因此没有被 TranSys 采用。

由于决策树在大小上更小、在计算上更简单、对管理员来说更透明，我们可以解决前面所述的现有基于深度强化学习的智能网络系统的重量级和不透明两大问题[2.3 节]。例如，在 AuTO 中将深度神经网络替换为决策树可以提高决策的实时性并因此提高性能[4.3 节]。同时，对 Pensieve 来说，这也让我们能够直接在移动设备或个人电脑上，在资源消耗大幅削减的情况下进行部署[4.4 节]。更进一步，因为 TranSys 的决策过程是透明的，我们还可对深度神经网络模型进行故障检测[4.5.1 节]和解释[4.5.2 节]。

3.3 数据集重采样方法

如图 3.1 所示，TranSys 首先运行训练好的模型，然后将其(s_t, a_t, r_t)元组收集起来，组成数据集$\mathcal{D}$。这背后的原因是：既然深度强化学习的损失函数和现有决策树算法的并不匹配，那我们就根据模仿学习过程中的损失函数对采集到的数据集进行重采样，这样我们便可以利用现有算法直接训练决策树。我们下面描述基于[13]的重采样方法：

在强化学习中，给定状态集合$\mathcal{S}$，决策集合$\mathcal{A}$，奖励函数$R(s)$，转移概率$p(s'|s,a)$，和策略$\pi: \mathcal{S} \to \mathcal{A}$，令：

$$V_t^{(\pi)}(s) = R(s) + \sum_{s' \in \mathcal{S}} p\big(s' \big| s, \pi(s)\big) V_{t+1}^{(\pi)}(s') \tag{3-1}$$

$$Q_t^{(\pi)}(s,a) = R(s) + \sum_{s' \in \mathcal{S}} p(s'|s,a) V_{t+1}^{(\pi)}(s') \tag{3-2}$$

分别为在有限时间（T步迭代）中的价值函数和 Q 函数，并有$V_T^{(\pi)} = 0$。价值函数代表着从状态s开始，沿用策略π情况下的预期的总的奖励。Q 函数则进一步指定了下一步的决策应为a。因此，在当前状态为s，学生策略为π的情况下，模仿学习的损失函数可被表示为：

$$\ell(s,\pi) = V^{(\pi^*)}(s) - Q^{(\pi^*)}\big(s, \pi(s)\big) \tag{3-3}$$

其中，π^*为老师（深度神经网络）的决策策略。由于学生策略π是在数据集重采样之后才能被训练出来，在这一阶段尚不知道，所以我们通过对公式(3-3)取上界来去除π的影响：

$$\ell(s,\pi) \le \tilde{\ell}(s) = V^{(\pi^*)}(s) - \min_{a' \in \mathcal{A}} Q^{(\pi^*)}(s,a') \tag{3-4}$$

为了将决策树和深度神经网络的损失函数相匹配，我们根据上面的损失函数的上界来对数据集$\mathcal{D}$重采样，以建立数据集$\mathcal{D}'$：

$$p(s,a) = K \cdot \tilde{\ell}(s) \cdot \mathbb{I}[(s,a) \in \mathcal{D}] \tag{3-5}$$

其中，K是归一化系数，$\mathbb{I}[(s,a) \in \mathcal{D}] \in \{0,1\}$指示状态-决策对$(s,a)$是否在数据集$\mathcal{D}$中。在这种情况下，直接在数据集$\mathcal{D}'$上面采用 CART 等决策树算法来训练决策树等价于以优化强化学习的损失函数为目标的训练过程[13]。对于含有基于 actor-critic[43]的强化学习智能体的网络系统[21][53][67]而言，价值函数是从神经网络

中计算而来，因此没有可用的 Q 函数。在这种情况下，我们将公式(3.4)中的 Q 函数替换为强化学习的熵表达形式[13][85]：

$$Q^{(\pi^*)}(s,a') = log\,\pi^*(s,a') \tag{3-6}$$

3.4 决策树训练方法

给定已经经过重采样的数据集$\mathcal{D}'$，我们接下来可以采用 CART 算法来训练决策树。决策树训练分为两个过程：决策树生成和代价复杂度剪枝（Cost Complexity Pruning，简称 CCP）。在第一阶段，我们根据数据集$\mathcal{D}'$中的数据，产生一个完全分开的决策树。在此时，决策树的大小远大于我们所需的大小。我们接下来采用代价复杂度剪枝来根据网络管理员的需求减少叶子节点的数量。

表 3.1　决策树生成算法

输入： 重采样过的数据集$\mathcal{D}' = \{(s,a)\}$

输出： 决策树$\hat{\pi}$

1 将决策树初始化为根节点$\mathcal{T} = \{root = \mathcal{D}'\}$

2 **while** 叶子节点没有完全分开 **do**

3 　**foreach** $n \in L(\mathcal{T})$且尚未计算过$\Delta_G(n)$ **do**

4 　　基尼增益$\Delta_G(n) = \max_i \max_c \left(\frac{N_1}{N} I_G(n_1) + \frac{N_2}{N} I_G(n_2) - I_G(n)\right)$

5 　选择叶子节点$n = \arg\max \Delta_G(n)$

6 　更新$\mathcal{T} \leftarrow \mathcal{T}$将$n$分割为$n_1$和$n_2$

3.4.1 决策树生成算法介绍

在决策树中，每个节点包含一些样本（也就是状态-决策对）。每个叶子节点根据所包含的样本中的主要类别来进行决策。每个内部节点会根据节点自身的分类准则将其样本分割到它的两个子节点中。如表 3.1 中的算法 1 所示，我们首先将决策树$\mathcal{T}$初始化为含有所有样本的一个根节点。我们接下来为叶子节点集合$L(\mathcal{T})$中的每个节点计算基尼不纯度：

$$I_G(n) = 1 - \sum_i p^2(i,n) \tag{3-7}$$

其中，$p(i,n)$是在节点n的所有样本中，决策i出现的频率。基尼不纯度是熵（$\sum_i p_i \log p_i$）的一个近似表示。$I_G(n)=0$表示节点n中的所有样本都有相同的决策（在同一类中）。一个较高的$I_G(n)$则表示当前节点中的样本在不同类别中较为平均。在这种情况下，在那个节点中的样本需要进一步分类到其叶子节点中去。对于叶子节点n而言，我们计算其基尼不纯度增益以及其最佳的分割变量s_i和分割点c。n_1和n_2是n根据上面的分割准则产生的两个叶子节点（第 4 行）。我们之后将具有最大增益的叶子节点进行分割并得到一棵新的决策树（第 5-6 行）。这一循环持续到所有叶子节点均被分割，也就是说$\forall n \in L(\mathcal{T}), I_G(n)=0$。

3.4.2 代价复杂度剪枝介绍

对于上面产生的决策树，由于在叶子节点上全部样本是被完全分开的，决策树的大小远比我们需要的要大。例如，我们的实验发现将从 Pensieve 中采集到的 16 万的样本完全分开需要 1000 个叶子节点。因此，我们需要对决策树进行剪枝。网络管理员可根据具体的部署情况，将叶子节点综述限制到Θ。我们下面介绍代价复杂度剪枝[33]，其为决策树的复杂度引入惩罚因子。代价复杂度剪枝所用的损失函数为：

$$C_\alpha(T)=C(T)+\alpha|T| \tag{3-8}$$

其中$|T|$是决策树T的叶子节点数目，$C(T)$是决策树T的预测准确率，α是用来平衡复杂度和准确度的一个参数。对于每个内部节点n而言，如果我们将n以下的全部子节点移除，仅仅包含n一个根节点的这棵子树的损失函数为：

$$C_\alpha(n)=C(n)+\alpha \tag{3-9}$$

而带有以n根节点的子树T_n的损失函数则为：

$$C_\alpha(T_n)=C(T_n)+\alpha|T_n| \tag{3-10}$$

因此我们有：

$$\alpha > g(n)=\frac{C(n)-C(T_n)}{|T_n|-1} \Leftrightarrow C_\alpha(T_n) > C_\alpha(n) \tag{3-11}$$

公式(3-11)说明，当$\alpha > g(n)$时，将子树T_n替换为它的根节点n可以降低损失函数。因此我们为每个内部节点都计算其对应的$g(n)$，并逐渐从 0 开始增加α，按序将子树修剪掉，直到叶子节点数目下降到Θ。这会产生一个满足网络管理员要求的决策树。

3.5 连续状态输出研究

图 3.3 分类树与回归树（左为分类树，右为回归树，图源 Wikipedia）

对于网络系统中的连续状态输出，我们采用回归树的设计来产生实数的输出[77]。与产生离散输出的分类树相比，回归树能够在训练时连续地更新其叶子节点上的实数输出值。简而言之，回归树将状态空间分割为几个部分，每个叶子节点代表一个部分。回归树之后用在对应叶子节点上所有样本的期望作为叶子节点的预测值。在每次分割时的优化目标变为最小化每个节点内部的均方预测误差。针对于本例而言，回归树训练时将公式(3.7)中的基尼不纯度替换为节点内部样本的方差：

$$I_S(n) = \sum_i (y_i - \overline{y_i})^2 \tag{3-12}$$

同时，当对回归树进行剪枝的时候，我们采用预测误差作为公式(3.8)中的代价函数。对于含有 actor-critic 的强化学习智能体，因为这里对于连续输出既没有 Q 函数也没有熵，我们将公式(3-6)中的熵替换为输出值的置信区间[68][79]。在这种情况下，3.2 节与 3.3 节中的算法便可以被扩展到网络系统中的连续输出变量的情况了。

第4章 实验验证与分析

在本节中，我们将 TranSys 部署于 2.2 节中介绍的两个较具有代表性的基于深度强化学习的智能网络系统：Pensieve[①]和 AuTO[②]上。我们通过评估这两个系统的性能来评估 TranSys 的效果。基于 TranSys，我们为他们分别建立了两个基于决策树的系统：TranSys-over-Pensieve (ToP)和 TranSys-over-AuTO (ToA)。我们首先在 4.1 节中介绍部署细节，并通过回答以下问题来评估 TranSys 的性能：

- **保真度。**和原有基于深度强化学习的系统相比，TranSys 会不会引入性能上的损失呢？我们发现 TranSys 带来的性能损失在两个应用中分别不超过 0.6%（ToP）和 1.8%（ToA）。ToP 和 ToA 可以精确地模仿深度神经网络的决策，并有远超过当前启发式方法的性能[4.2 节]。
- **决策延迟。**TranSys 能够有效减少原有深度强化学习模型的决策延迟吗？我们的实验结果表明，ToP 和 ToA 两个系统的决策延迟都能够减少一到两个数量级，这同样也带来了显著的性能优势[4.3 节]。
- **资源消耗。**TranSys 能够有效减少网络设备上的资源消耗吗？实验证明，和原有基于深度强化学习的模型相比，ToP 可以将自适应比特率调整算法的额外页面加载时间减少到原来 156 分之一，实时内存利用减少到原来的 6.6 分之一[4.4 节]。
- **透明性。**TranSys 能增加决策过程的透明性并帮助网络管理员来管理智能网络系统吗？我们展示了 TranSys 带来的透明性的两个应用实例：检测基于深度强化学习的网络系统中的故障，和通过解释深度神经网络的结果来指导未来系统设计[4.5 节]。
- **代价。**网络管理员如果想要在实际中部署 TranSys，需要付出多少额外代价？我们的实验结果表明，将训练好的深度神经网络转换为决策树只需要几乎可以忽略的线下计算时间（小于一分钟）。同时，网络管理员不需要仔细调整 TranSys 的参数：TranSys 对超参数比较鲁棒[4.6 节]。

① https://github.com/hongzimao/pensieve

② https://bitbucket.org/JustinasLingys/auto_sigcomm2018

4.1 系统实现

图 4.1 实验环境

4.1.1 Pensieve 和 ToP 的实现与部署

除非特别说明，我们采用了和 Pensieve 一样的一段 193 秒长的测试视频。视频块大小、比特率分别被设置为 4 秒和{300, 750, 1200, 1850, 2850, 4300}kbps（分别代表{240, 360, 480, 720, 1080, 1440}p 的视频分辨率）。真实网络流量数据包括由 Pensieve 作者提供的 250 段挪威 3G-HSDPA 网络数据[65]和 205 段美国广域网 FCC 数据[26]。我们设立了和 Pensieve 相同的测试环境：用一个 Google Chrome 浏览器作为视频播放客户端、一个在同一机器上的 Apache 视频服务器。我们采用 Mahimahi 仿真器[59]来仿真网络情况，并将往返时间设置为 80ms。我们参照[53]

中的指示，并采用 Tensorflow.js[4]将神经网络集成到 JavaScript 中以使得能够直接在浏览器中运行 Pensieve。

我们采用线性 QoE 作为优化目标：

$$QoE = \frac{1}{N}\left(\sum_{n=1}^{N} R_n - \mu \sum_{n=1}^{N} T_n - \sum_{n=1}^{N-1} |R_{n+1} - R_n|\right) \tag{4-1}$$

在总共N块中的第n块时，R_n和T_n代表比特率和卡顿时间。公式(4-1)中的三项分别代指视频质量、卡顿惩罚和平滑度惩罚。采用其他种类的 QoE 产生的结果是相似的。在我们的实验中，我们采用[53]提供的经过精调后的模型。

对 ToP 而言，我们采用 scikit-learn[62]来实现决策树训练，并对其进行修改以支持代价复杂度剪枝。我们在一台装配有一块 Intel Core i7-8700 CPU（6 个物理核）和一块 Nvidia Titan Xp GPU 上训练决策树。我们将重采样后的样本数量设置为 20 万，并随机将其分割为训练集（80%）和测试集（20%）。经过几次经验性地尝试，我们将叶子节点数量设置为 200。我们采用 sklearn-porter[58]将决策树转换为 JavaScript 代码。

我们还采用了近年来开发的几个自适应比特率调整算法作为基线：

- BB[40]：一个基于缓冲的解决方案。其根据缓冲区空闲情况根据分段线性函数选择比特率。
- RB[53]：一个基于带宽的解决方案。其根据前五块的调和平均预测带宽，并选择允许范围内的最高比特率。
- FESTIVE[42]：一个加强的基于带宽的方案。其能够在公平性和效率上取得平衡。
- BOLA[71]：一个加强的基于缓冲的方案。其采用了李雅普诺夫优化进行求解。
- rMPC[82]：一个根据带宽和缓冲来综合优化的方案。

我们接下来将其迁移进 dash.js 客户端[2]。我们采用[53]提供的基线实现方案。

4.1.2 AuTO 和 ToA 的实现与部署

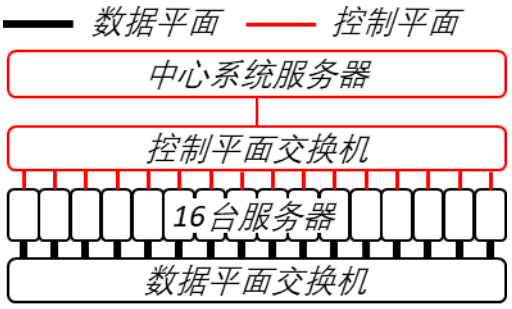


图 4.2 AuTO 部署拓扑结构

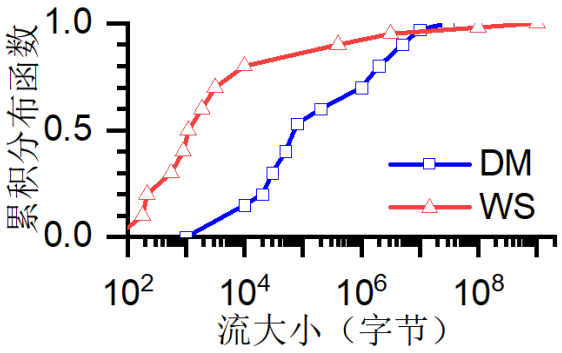


图 4.3 AuTO 部署中采用的数据流量特征

如图 4.2 所示，我们采用了和 AuTO 一样的 16 服务器拓扑。数据流量和 AuTO 一样，包含一组网络搜索（Web Search，简称 WS）流量[36]和一组数据挖掘（Data Mining，简称 DM）流量[8]，如图 4.3 所示。由于作者没能够提供预训练模型，我们按照[21]中的指示将神经网络训练了 8 个小时。我们使用两台 H3C-S6300 的 48 端口交换机。所有的其他配置（如：链路容量、链路负载、神经网络结构）均与 AuTO 一致。我们将 ToA-lRLA 和 ToA-sRLA 的叶子结点个数设置为 2000。这是因为 ToA-lRLA（143 个状态）和 ToA-sRLA（700 个状态）的状态空间要比 ToP（25 个状态）大得多。进一步地，我们在 4.6.2 节中说明了这一参数的鲁棒性。关于决策树的其他部署细节均与 ToP 一致。

4.2 转换系统的保真度评估

我们在两个层面上说明 TranSys 的保真度：我们首先说明基于 TranSys 的系统在应用级别性能上（对 Pensieve 来说是 QoE，对 AuTO 来说则是 FCT）和原有

基于深度强化学习的系统是相当的。然后我们深入到决策树和深度神经网络做的每个决策中去。我们发现决策树可以高准确度、低误差地模仿深度神经网络。

4.2.1 应用级别性能评估

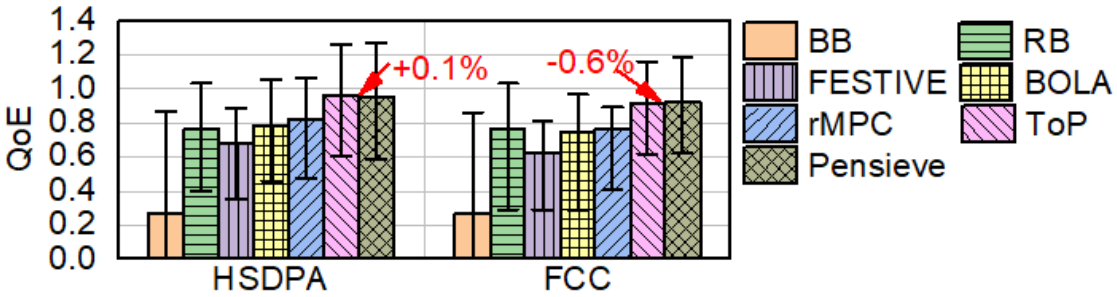


图 4.4 两组数据流量上的 QoE。误差线代表 25 百分位和 75 百分位。

对 ToP，正如我们在 4.1.1 节中所言，我们将 ToP 的 QoE 和 Pensieve 以及其他基线方案在两组数据流量上进行对比，将 QoE 结果的平均值和 25 百分位、75 百分位结果展示在图 4.4 中。ToP 和 Pensieve 的平均 QoE 的差别在两组数据流量上分别为+0.1%和-0.6%。同时类似于 Pensieve，ToP 的性能超过其他基线至少 14.4%。

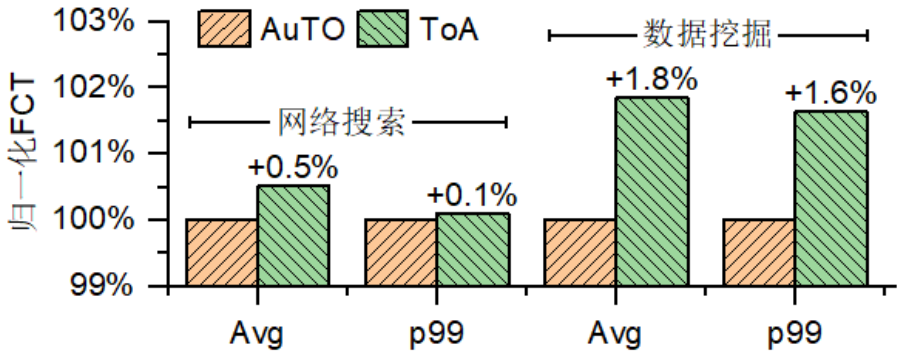


图 4.5 被 AuTO 结果归一化后的流完成时间

对 ToA 而言，我们根据[21]中的实验，测量了流完成时间的平均值的 99 百分位数据，如图 4.5 所示。和 AuTO 相比，ToA 能够将性能损失控制在 0.51%和 1.84%之内。相比之下，可供参考的是 AuTO 可以最多降低 48.14%的流完成时间（在[21]

中给了详细分析）。因此，ToA 带来的 1%左右的性能损失是几乎可以忽略的。总之，TranSys 能够只利用决策树就有和深度神经网络相当的性能。

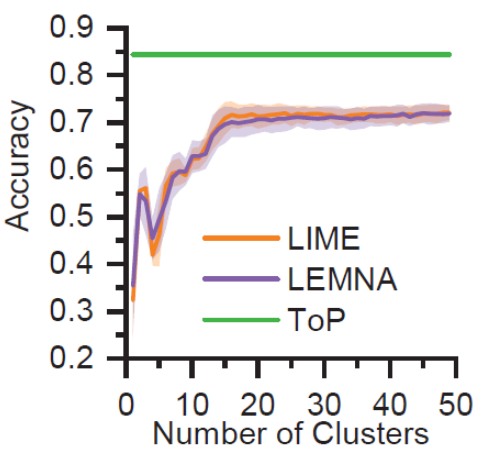
4.2.2 单个决策准确度评估

进一步地，我们想知道 TranSys 能够保持性能的原因。我们测量了决策树和原有深度神经网络的输出的准确度和方均根误差。我们将 TranSys 具体应用到三个深度神经网络中来进行测量比较。作为基线，我们还实现了下面两个可以提供更简单模型的可解释性方法进行对比：

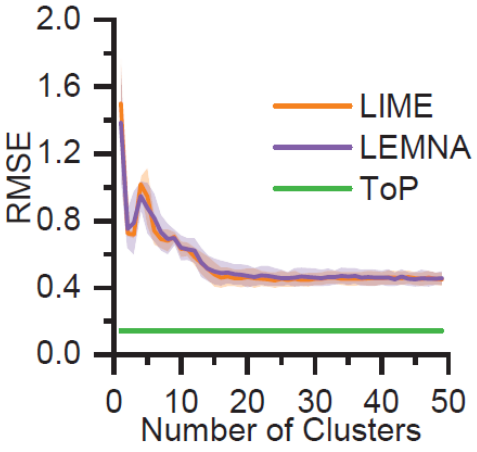
- LIME[63]通过线性回归来预测和解释。
- LEMNA[37]采用混合回归来预测局部非线性。

由于上述两种方法都是为局部预测而设计，而 TranSys 是为全局预测而设计，为使比较更加公平，我们通过下面的方式来实现基线方案：在训练时，我们采用 k-means 聚类[51]将样本分成k组。我们之后在每组内训练 LIME 和 LEMNA。当为一个新的样本进行预测时，我们首先找到新样本所属的最近的一组，然后采用该组结果进行预测。我们改变k从 1 到 50，并重复实验 100 次以消除随机性。结果如图 4.6 所示。ToP 和 ToA 不需要被聚类，因此在图中是一条常数线。

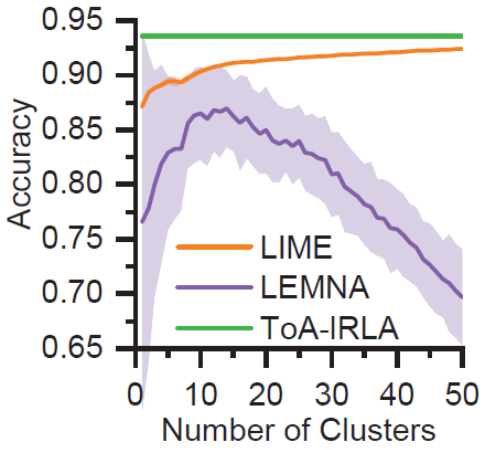
从图 4.6(a)和图 4.6(c)中可看出，与原有深度神经网络相比，ToP 和 ToA-lRLA 分别实现了 84.3%和 93.6%的高精度。由于在流调度领域内的最先进水平的决策逻辑[9][11]比自适应比特率调整算法（如：随机优化[82]、李雅普诺夫优化[71]）要简单得多，ToA-lRLA 的准确度比 ToP 要高。图 4.6(b)、图 4.6(d)、图 4.6(e)中的低误差说明即使决策树做出了和神经网络不一致的决策，二者差的也不太远，也就不会带来严重的性能下降。决策时能够精确模仿深度神经网络使得 4.2.1 节中的可忽略的应用级别性能损失成为可能。同时，TranSys 的准确度和方均根误差远比 LIME 和 LEMNA 要好。因此我们在 3.1 节中的设计选择也得到了印证：决策树有更丰富的表达能力，并更适合于网络系统。LEMNA 的性能在 ToA 的两个智能体上并不稳定，这是因为 AuTO 的状态高度集中在几处地方，便影响了 LEMNA 期望优化迭代的性能[37]。



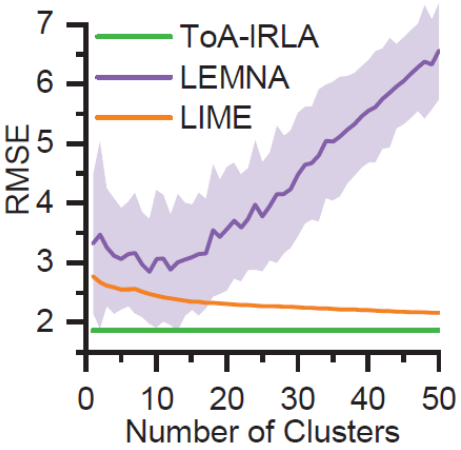
(a) 准确度（ToP）



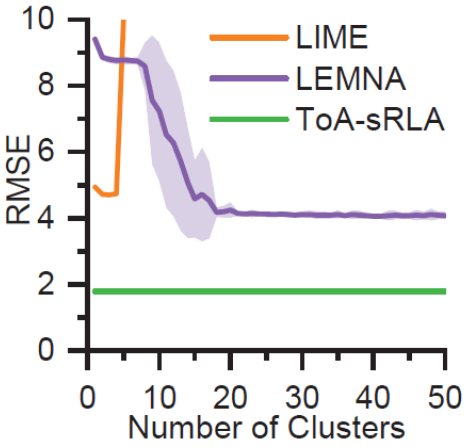
(b) 方均根误差（ToP）



(c) 准确度（ToA-IRLA）



(d) 方均根误差（ToA-IRLA）



(e) 方均根误差（ToA-sRLA）

图 4.6　ToP 和 ToA 的保真度。阴影代表平均值外一个标准差。sRLA 产生实数预测结果，因此没有准确度。当聚类数量大于等于 5 时，LIME 在 sRLA 上的结果发散。

4.3 转换后系统的决策延迟评估

我同时还测量了 ToP 与 ToA 两个网络系统与不同方法相比的决策延迟。对于 ToP 来说，由于决策延迟依赖于底层设备的计算能力，除了我们在 4.1.1 节中介绍的个人电脑之外，我们还测量了在装有高通骁龙 710 处理器的手机（Android 8.1.0）上的决策延迟。决策延迟的测量依赖于 JavaScript 中的 performance.now API。如图 4.7 所示，在应用级别性能几乎相同的情况下（图 4.4），与 Pensieve 相比，ToP 能够将这一延迟在两个设备上分别减少到原来的 34.7 分之一和 36.6 分之一。这个启发式方法在同一数量级上（毫秒级）。

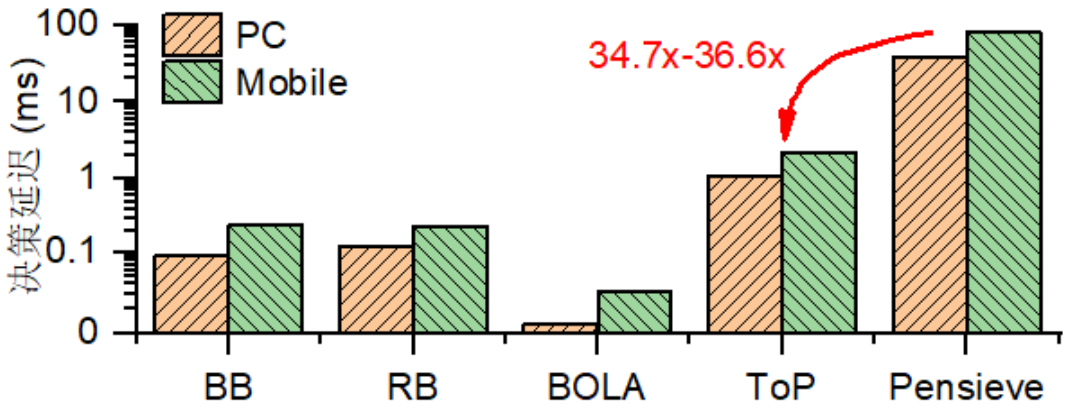
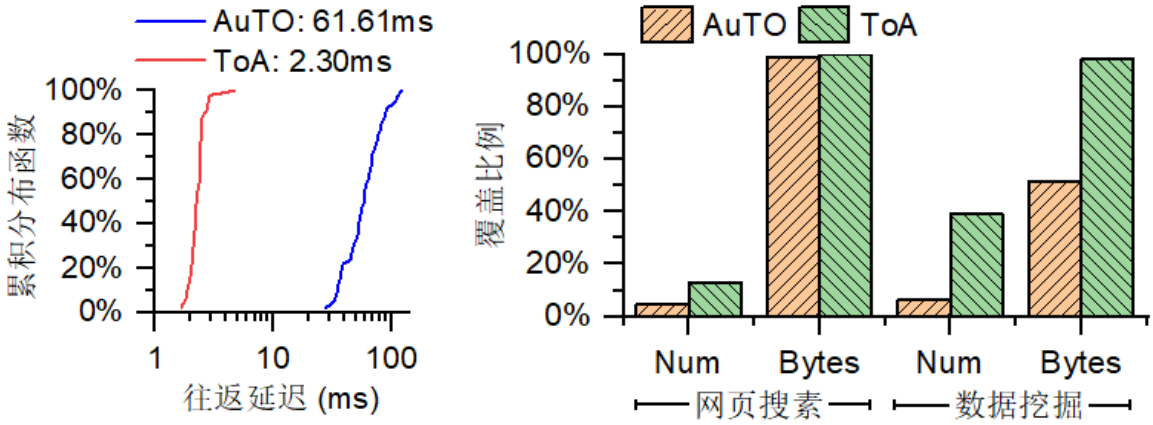


图 4.7　不同端设备上的决策延迟

由于其逐流决策延迟在 100ms 级别，短流在这一过程中等不及就会完成，AuTO 为长流和短流设计了不同的策略。将深度神经网络转换为决策树会缩短决策延迟，也就使得我们能够精确地为更多的流做出逐流决策。如图 4.8(a)所示，当把深度神经网络替换成决策树后，决策延迟可以被缩短至原来的 26.8 分之一至平均 2.30ms。在这种情况下，对于网页搜索流量分布，基于 TranSys 的系统能覆盖 39%的流和 98%的字节。即使对于数据挖掘流量，根据我们的实验，ToA 也能覆盖 6%和 52%，如图 4.8(b)所示。在这种情况下，我们可以为更多的流进行逐流控制：不仅仅是长流，现在我们对很大一部分中等大小流也可以做逐流控制。



(a) 延迟　　(b) 平均延迟的覆盖范围

图 4.8　流服务器向强化学习服务器发送状态到收到决策的延迟

4.3.1　低决策延迟的性能收益分析

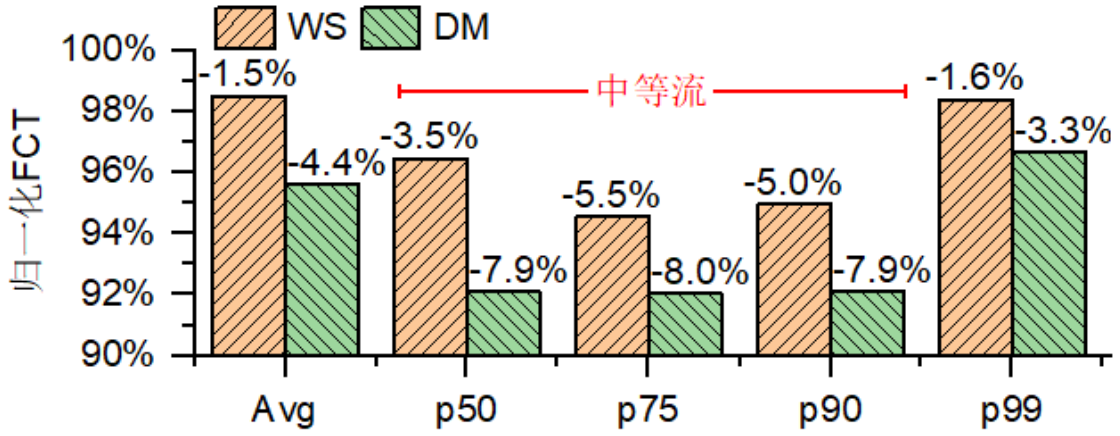


图 4.9　当流服务器将所有流上报时，被 AuTO 归一化的流完成时间。更低的流完成时间（< 100%）代表更好的性能

受 ToA 的较好的实验结果所驱动，一个直觉是增加经过优化后的主流控制范围会提升整体性能。我们因此基于 ToA 实现了一个高覆盖范围的逐流控制系统的原型：我们遵循 AuTO 中关于 sRLA 的设计方案，并仍然采用其结果来控制优先级队列。然而，与 4.2.1 节中 AuTO 仅向上汇报最后一个队列的数据流不同，我们将所有队列里的数据流都上报数据平面并用决策树来实现快速实时决策。短流在决策返回之前会完成，因此仍然遵循 AuTO 中的多级反馈队列设计。然而，由于等待决策的往返时间被缩短了，更多的中等流能够收到有效的调度决策，并因此

遵循经过优化后的逐流决策。我们测量了不同百分位下的经过归一化的流完成时间，结果如图 4.9 所示。尽管决策树在训练的过程中没有遇见过怎样处理中等流，他仍能够将平均性能在网页搜索和数据挖掘两个数据集上提升 1.5%和 4.4%。同时，我们在中等流（从 50 百分位到 90 百分位）上能观察到多达 8%的性能提升。这说明中等流在享受逐流精确调度带来的收益。数据挖掘流量上的性能提升比在网页搜索上更加明显，这是因为其覆盖范围更大，如图 4.8(b)所示。

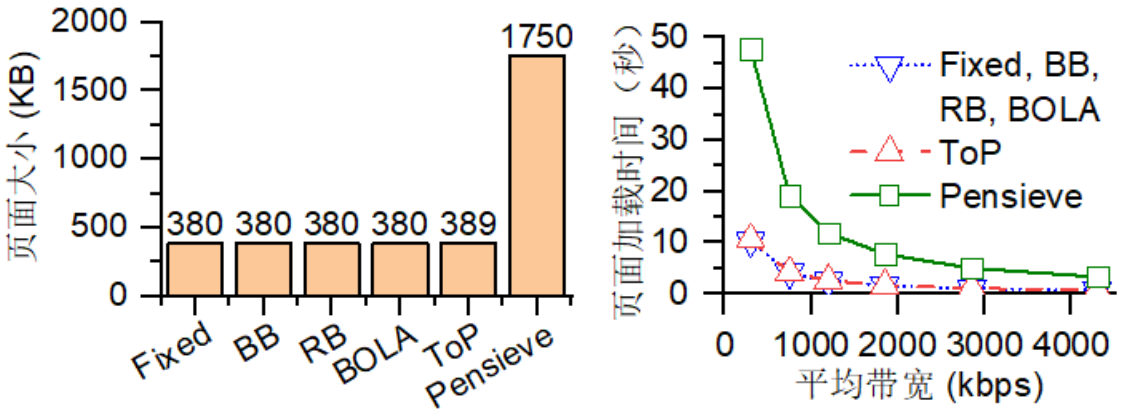
4.3.2 低决策延迟的部署收益分析

另一个将深度神经网络转换为决策树带来的收益是他们可以被部署到编程能力受限的设备上。对于 AuTO 中的神经网络来说，由于这里有大量的神经元参数和许多复杂的计算（如：浮点数乘除法、sigmoid 函数），他们很难用诸如 P4[16]的低级语言实现。相比之下，由于决策树的内部逻辑仅由 if-else 分支和比较来实现，决策树能够通过将分支逻辑转换为查找-动作表来实现。这使得我们能够将决策树部署到可编程网卡[32]或可编程交换机[17]上，并将整个决策过程卸载到数据平面，进一步扩大逐流调度的覆盖范围到所有流。我们通过将决策树实现到一个支持 P4 编程的 Netronome NFP-4000 可编程智能网卡[60]上来证明其可行性。我们部署 ToA-lRLA 决策树并测量其决策延迟。在 NFP-4000 上的实验结果表明平均决策延迟是 $9.37\mu s$。这一延迟在高速可编程交换机上将会被进一步缩减。我们将在现有最高水平可编程交换机（如：Barefoot Tofino[61]）上的实现及其和其他基线方案[9][57]的对比作为下一步工作。

4.4 转换系统的资源消耗评估

对 AuTO 来说，由于深度神经网络最初是被部署在远程控制器上，尽管 ToA 将会通过将复杂的神经网络转换为决策树来减少其运行时资源消耗，但一般来说，远端控制器上的资源并不是一个严重的问题。因此我们在这里只评估 ToP 的资源消耗。由于 JavaScript 中还有其他的函数功能（如：播放器初始化），我们将自适应比特率调整算法与固定比特率调整算法相比。固定比特率算法总是选择最低的比特率（300kbps）。在这种情况下，我们测量自适应比特率调整算法的净代价。我们评估这些自适应比特率调整算法的初始页面加载时间[29]和运行时内存消耗。

4.4.1 页面加载时间评估



(a) 页面大小　　(b) 页面加载时间

图 4.10　不同自适应比特率调整算法的 HTML 页面

页面加载时间是用户下载包含视频播放器的 HTML 页面所需的时间[29][82]。如果页面大小过大，在视频开始播放之前，用户需要等待较长的时间。这可能会使得一部分用户丧失耐心并离开页面。对于 Pensieve 来说，自适应比特率调整算法所需要的模型需要在视频开始播放之前下载下来。我们用 Grunt.js[3]来压缩 JavaScript 代码，并测量在视频开始播放之前需要下载的页面大小。如图 4.10(a)所示，由于 Fixed，BB，RB，BOLA 的处理逻辑都比较简单，他们的页面大小差不多。由于深度神经网络的笨重性，Pensieve 将页面大小增加了 361%，从而导致了额外页面加载时间急剧恶化。与此相对，ToP 拥有和启发式算法相差不多的页面大小（只增加了 2%）。如图 4.10(b)所示，当我们将带宽设定为 1200kbps（这比 HSDPA 数据集中 54%的平均带宽都要高[65]）时，自适应比特率调整算法引入的额外页面加载时间被缩短到了原来的 156 分之一：Pensieve 会引入 9.36 秒的额外加载时间（$\frac{(1750-380)KB}{1200\text{kbps}}$），而 ToP 只增加了 60ms（$\frac{(389-380)KB}{1200\text{kbps}}$）的页面加载时间。

4.4.2 运行时内存消耗评估

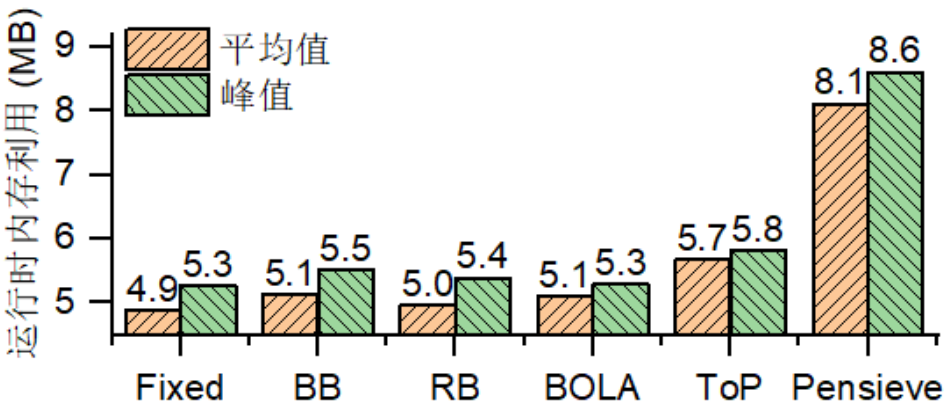


图 4.11 运行时 JavaScript 内存消耗。固定比特率算法的内存消耗是由 JavaScript 中的其他功能造成的。

我们接下来测量了运行时内存的平均值和峰值。我们利用 Chrome DevTools[1] 中的 memory API 添加了一个外部模块，以周期性地测量 JavaScript 堆内存。如图 4.11 所示，由于神经网络前向传播的计算复杂度，和其他自适应比特率调整算法相比，Pensieve 需要消耗更多的资源（平均多 3.2MB）。与此相对，ToP 的额外内存消耗平均值被减少到了 Pensieve 的四分之一，峰值被减少到了 Pensieve 的 6.6 分之一。ToP 的内存消耗和其他启发式方法在同一级别。

4.5 转换系统的透明性评估

我们展示了利用 TranSys 所提供的透明性的两个应用实例：一个用 TranSys 来检测 Pensieve 做出次优决策的故障；和一个 Pensieve 决策逻辑的初步分析，并为未来自适应比特率调整算法设计提供思路。

4.5.1 故障检测用例：消失的比特率

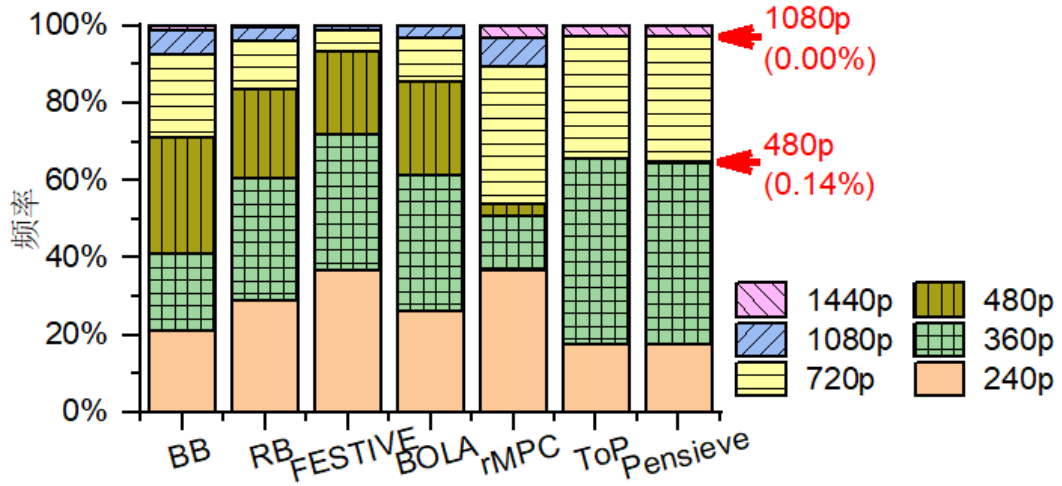


图 4.12 比特率选择的频率（HSDPA 数据集，12250 次决策）

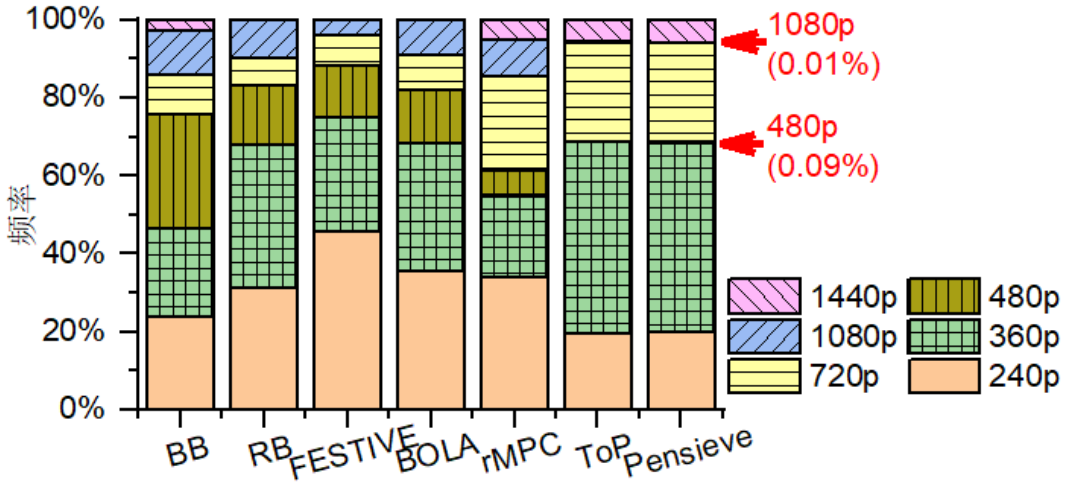


图 4.13 比特率选择的频率（FCC 数据集，10045 次决策）

当在经过重采样的数据集$\mathcal{D}'$上训练 ToP 时，我们发现有些比特率很难被 Pensieve 选中。在 4.2.1 节的实验中不同比特率被选择的频率如图 4.12 和图 4.13 所示。一方面，ToP 的结果和 Pensieve 几乎一致，这进一步印证了 ToP 的高保真度。另一方面，480p 和 1080p 几乎没有被 Pensieve 选中，这激起了我们的兴趣：根据网络情况的不同，最高或最低的比特率或许可能不怎么被选中，但这两个比特率都是中间比特率，并不能用这种原因来解释。

为了进一步探索这背后的原因，我们又对 Pensieve 做了如下实验：我们在一系列带宽固定在 300kbps 到 4500kbps 的链路上测试 Pensieve，并收集其决策结果。

由于[53]中所用的视频样本太短以至于不能明确表达这一问题，我们将视频替换为一个 1000 秒（共 251 视频块）的视频，并保持其他配置和原有实验相同。

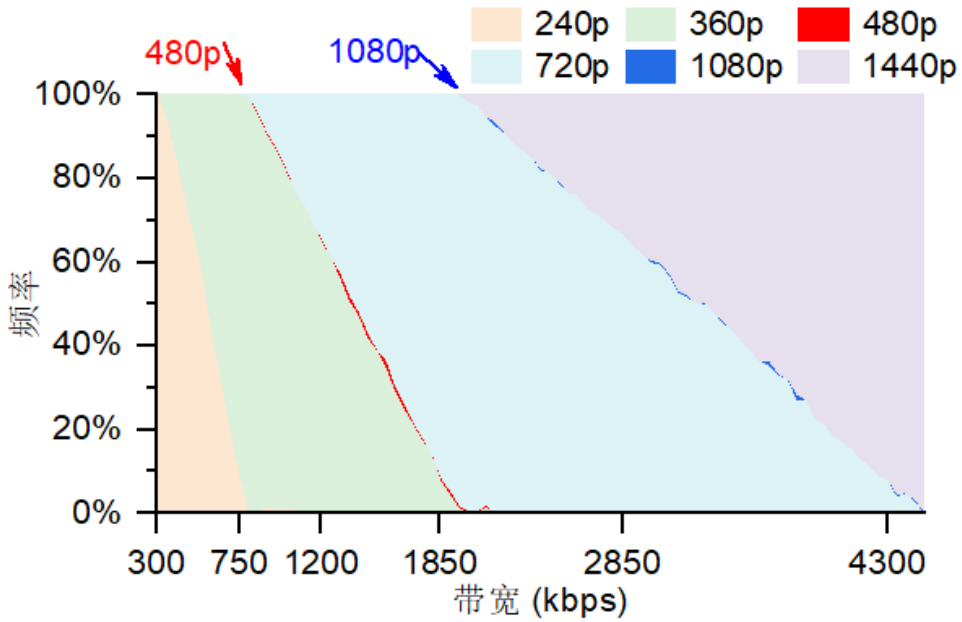


图 4.14　在固定带宽链路上，比特率被 Pensieve 选择的频率

如图 4.14 所示，480p 和 1080p 仍旧不被 Pensieve 所选择。例如，当链路带宽在 3000kbps 左右时，最优的决策应该是一直选择 1080p 的分辨率，正如在 2000kbps 左右 Pensieve 一直选择 720p 一样。这个带宽比 1080p 的比特率（2850kbps）略高一些，这是因为有效吞吐和带宽之间的差别。然而，在这种情况下，Pensieve 的决策中只有 0.4%是 1080p，其他的选择在 720p 和 1440p 之间摇摆。

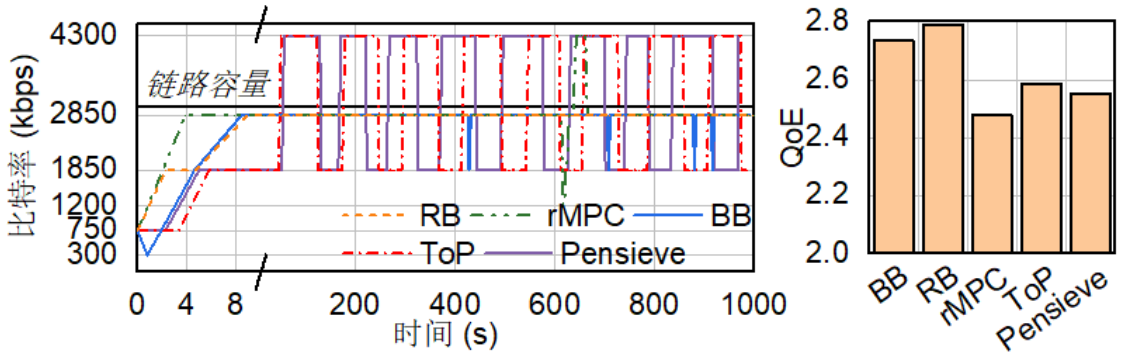


图 4.15　在 3000kbps 链路上比特率的选择。BB、RB 和 rMPC 收敛到 280kbps，而 Pensieve 和 ToP 则在 1850kbps 与 4300kbps 之间波动。

我们进一步在图 4.15 中展示了在 3000kbps 链路上随着时间变化的比特率选择情况。Pensieve 的决策在 720p（1850kbps）和 1440p（4300kbps）之间来回摆动。ToP 保真地模仿到了这一点，但这并不是最优的。相比之下，其他基线算法知道什么是较优的选择策略，并将其比特率决策固定在 1080p 上。因此 BB 和 RB 都有着比 Pensieve 和 ToP 更高的用户体验[①]。当链路带宽被固定在 1200kbps（480p）上时，可观察到类似的现象。进一步研究表明，Pensieve 在两个决策上都没有足够的信心[附录 B]。我们意识到 Pensieve 这里可能有问题。

这一问题或许由深度强化学习的训练机制导致。Pensieve 采用了著名的策略梯度法来训练。在每一步中，其智能体希望强化能够产生更高奖励的特定决策。在这种情况下，当智能体发现六个决策中的四个就可以获得一个相对较好的奖励，它便会通过不断选择这些决策来强化这一发现，并最终导致抛弃剩下的两个决策。用更少的决策来做决定可以为智能体带来更高的信心，但也会让其收敛到一个局部最优值，正如本例中的那样。

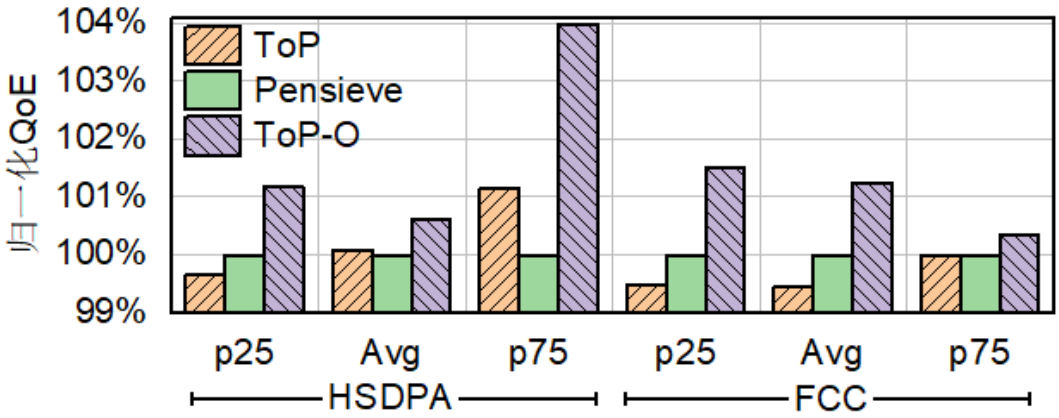


图 4.16　Pensieve 归一化的过采样 ToP（ToP-O）的用户体验

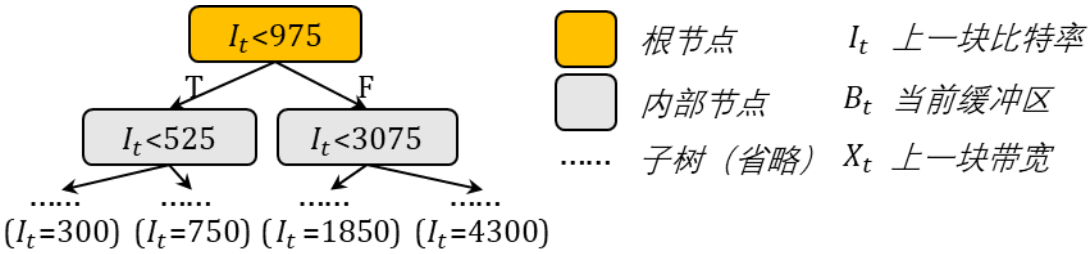
在没有 TranSys 的情况下，由于神经网络内部的决策过程对网络管理员而言并不透明，我们能做的唯一事情便是鼓励深度强化学习在训练阶段的探索，并花费数小时到数天来重新训练模型。然而，将深度神经网络转换为决策树使得我们能够通过调整其内部结构来手动解决这一问题。由于训练决策树的数据集是高度

① rMPC 的性能比 Pensieve 要差。这是因为它收敛得太快，以至于没有为视频块大小可能的波动预留足够的缓冲区[附录 B]。

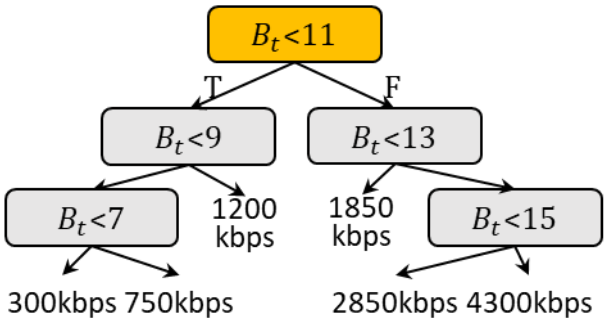
不平衡的，一个直接而又简单的解决方案就是将消失的比特率进行过采样：我们将 480p 过采样 10 倍、将 1080p 过采样 50 倍，使得他们的出现频率在 1%量级。注意到，过采样与第 3 章中以高保真度为目的的设计有冲突，会损害 ToP 的保真度。然而，由于 Pensieve 的策略并不是最优的，如果我们想找到它的问题并改善其性能，我们不能够盲目模仿 Pensieve 的决策。我们将过采样后的结果用 Pensieve 的用户体验进行归一化，并展示在两个数据流量集上的结果如图 4.16 所示。过采样后的 ToP（ToP-O）仅仅用决策树便超出神经网络的性能多达 4%。由于决策过程对我们而言是完全透明的，更精细化的设计（如：采用不均衡决策树[7][48]来让决策更加均衡）或能进一步提升性能。我们将其留作下一步工作。

4.5.2 深入可解释性：决策树为我们带来的新发现

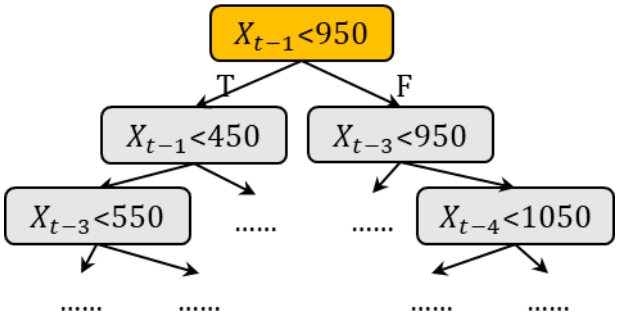
TranSys 带来的另一个好处便是可解释性本身。深入研究决策树的结构能够帮助我们了解深度强化学习模型学到和发现了什么新的知识。我们将从 Pensieve、BB 和 RB 中提取到的决策树的一部分展示如图 4.17 所示。如图 4.17(a)所示，在根节点上，Pensieve 首先根据其上一块的比特率来进行分类。如我们在 4.5.1 节中所讨论，Pensieve 没有考虑 1200kbps 和 2850kbps，因此没有为他们有特定的分支。接下来 Pensieve 为每个比特率产生定制化的策略（未在图中展示）。这和现有的 BB 或 RB 算法[42]不同。如图 4.17(b)和图 4.17(c)所示，当前启发式算法的结构要么受当前缓冲区空闲情况、要么受前面几块的带宽情况所控制。因此通过将 Pensieve 中的深度神经网络转换为决策树，我们发现了自适应比特率调整算法设计的一个新的可能的准则：上一块比特率中所蕴含的信息可能比我们预期的要更多。这一观察为设计启发式自适应比特率调整算法设计带来了新思路：网络管理员可以为不同的上一块比特率选择结果设计定制化的算法。我们在未来计划利用决策树，提供更多的基于深度强化学习的智能网络系统决策过程的解释。



(a) Pensieve



(b) BB



(c) RB

图 4.17 自适应比特率调整算法的决策树表达（最上面几层）

4.6 系统部署代价评估

最后，我们讨论可能会影响 TranSys 大规模实际部署的几个因素。我们首先证明，对于现有基于深度强化学习的智能网络系统而言，TranSys 产生决策树的线下计算代价是可忽略的。我们接下来展示，TranSys 对于不同超参数设定是较为鲁棒的。因此网络管理员不需要花费大量精力去调整参数。

4.6.1 线下计算时间评估

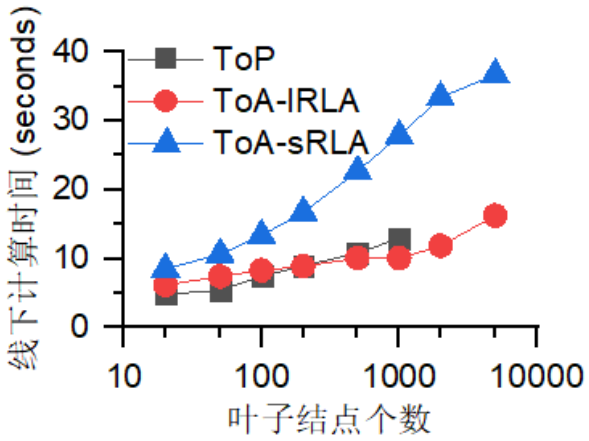


图 4.18 TranSys 线下计算代价

我们进一步检查了 TranSys 在决策树提取中的计算代价。我们在我们的测试床上测量了 ToP、ToA-lRLA、ToA-sRLA 在不同叶子节点数量下的决策树计算时间。由于 ToP 的决策空间（6 种决策）远小于 ToA-lRLA（108 种决策）和 ToA-sRLA（实数决策），当大约到 1000 个叶子节点时，ToP 的决策树便已被完全分开。因此在不扩大数据集的情况下，我们难以为 ToP 产生具有功能更多叶子节点的决策树，如图 4.18 所示，即使我们将叶子节点数量设置为 5000，这一计算时间仍然小于一分钟。由于决策树是在线下深度神经网络训练后进行，和深度神经网络训练相比（如：Pensieve 在有 16 个并行训练智能体下需要至少 4 小时[53]，AuTO 需要 8 小时[21]），这一额外的时间是可忽略的。TranSys 能够以可忽略的计算代价将深度神经网络转换为决策树。

4.6.2 敏感性分析

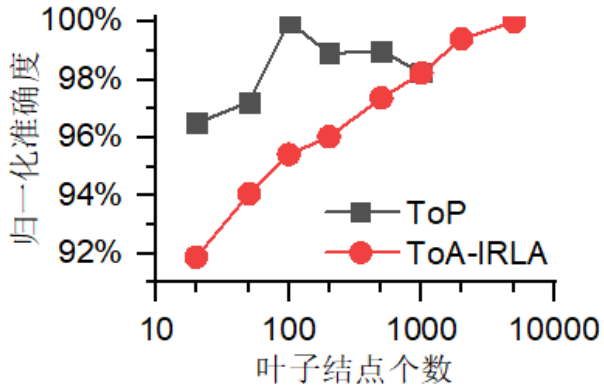


图 4.19　预测准确度对叶子结点个数的敏感度。结果对每个曲线的最优值做了归一化

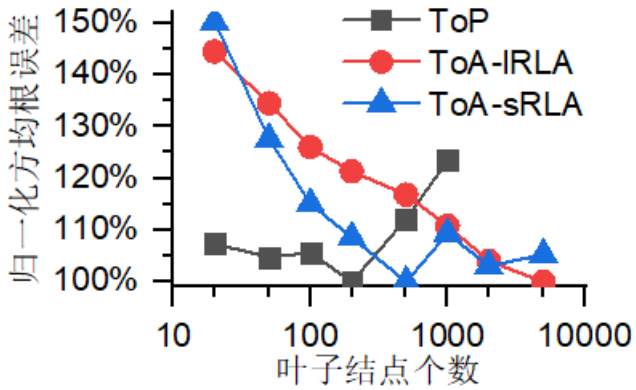


图 4.20　方均根误差对叶子结点个数的敏感度。结果对每个曲线的最优值做了归一化

在本小节中，我们测试 TranSys 对于超参数的敏感性。从第 3 章中可看出，TranSys 方法中对结果有潜在影响的超参数即为叶子结点个数：叶子节点数量越多，所产生的模型的资源消耗越多、决策延迟越长。同时，叶子节点数量越少，模型的保真度可能下降。

为测试 TranSys 对于不同叶子节点数量的鲁棒性，我们将叶子节点数量从 20 变到 5000，并测量其在 4.2 节中的三个智能体（ToP、ToA-lRLA、ToA-sRLA）的准确度和方均根误差。结果如图 4.19 和图 4.20 所示。对于 ToP，从 20 个叶子节点到 5000 个叶子节点的准确度和方均根误差都比中的 LIME 和 LEMNA 的最好结果要好。ToA-lRLA 和 ToA-sRLA 在从 200 叶子节点到 5000 叶子节点的很宽的一个区域内都要比 LIME 和 LEMNA 的最优结果要好。这说明网络管理员不需要在调整超参数上面花费许多时间：参数设定容许的范围很宽。

第5章 讨 论

5.1 可扩展性

在本文中，我们展示了 TranSys 在两个第 4 章种所介绍的基于深度强化学习的智能网络系统的有效性。然而，正如我们在 2.3 节中所说，两个系统都在应用深度神经网络的初级阶段：他们的隐藏层数目都小于 10。尽管如此，由于更深的神经网络的表达能力和较浅的神经网络相差不多[10]①，也就是说其也和决策树接近[3.1 节]。因此，即便随着神经网络层数的加深，其表达能力 TranSys 可被扩展到更深的神经网络中。在这种情况下，决策树能够高保真地去表达更深的神经网络所蕴含的策略。

当然，由于目前基于深度强化学习的智能网络系统中网络层数较浅，上述推断难以得到直接的印证。当有更多层的基于深度强化学习的智能网络系统出现时，我们将进一步测试 TranSys 在那上面的效果。

5.2 应用场景

TranSys 的设计目标是提出一个可适用于许多基于深度强化学习的智能网络系统的方法。我们通过在第 4 章介绍的两个较具有代表性的系统来说明了这一点。然而，当应用 TranSys 到不同场景下的网络系统时，我们仍然需要进行一些修改。例如，4.5.1 节中解决的 Pensieve 的问题可能不会被 AuTO 所遇到。

随着机器学习，尤其是深度强化学习在网络系统中的应用越来越多，TranSys 可被部署的场景也渐渐变多。在未来，我们将集成更多场景下的网络系统（如：低能耗物联网系统、任务调度[44][52][54]等）到 TranSys 中。

① 更深的神经网络更易于训练、收敛，因此被应用的更加广泛。

第6章 结论与展望

6.1 研究情况总结

在本文中，我们提出 TranSys——一种通过将深度神经网络转换为轻量级且透明的模型，从而将基于深度强化学习的智能网络系统在大规模部署中实用化的方法。在数据集重采样和回归树的帮助下，TranSys 能够高保真地将基于深度强化学习的网络系统中的深度神经网络转换为决策树。我们将 TranSys 应用于两个较具有代表性的基于深度强化学习的网络系统。试验结果表明，和原有系统相比，基于 TranSys 的系统有保真度高、决策延迟短、资源消耗低和透明度高等优点，同时还可以带来额外的性能收益。本工作未来计划开源发布，以便于提升工作的可重复性。

6.2 未来工作改进

同时，本文的研究还存在一定的不足，将智能网络系统实用化还有许多问题亟待解决。作者未来计划从以下几方面继续完成这一工作。

公平性。在现有工作中，网络系统的公平性受到了很多关注。例如，在集群任务调度[86]或拥塞控制[87]中，设计者通常通过显式的设计来确保系统的公平性。网络管理员需要确保，对于不同用户其应该尽量提供公平的服务方案。然而，最近有研究表明，深度神经网络有时可能会通过牺牲公平性的方法来换取全局性能的提高[88]。另外，由于这一决策过程是不透明的，网络管理员很难确保采用智能网络系统不会削弱用户的公平性。将基于深度神经网络的策略转换为决策树或许可以在某种程度上尝试解决这一问题，但仍需要深入研究。

动态性。网络系统通常需要运行时对其策略进行动态更新（如：交换机上可能需要更新流表）。然而，由于深度神经网络的决策是通过成千上万的神经元共同做出的，基于深度神经网络的系统很难进行动态调整[89]。例如，如果我们将基于深度强化学习的智能流调度系统[4.3.2 节]部署到交换机中，像直接调整启发式方法那样去调整神经网络的决策边界是很具有挑战性的。TranSys 系统带来的透明

性或许可以在一定程度上明确决策边界，但当随着树节点数的增长，我们仍然需要人力来手工对其进行调整。

安全性。对于网络系统来说，安全性也是非常关键的一个问题。[84]的作者提供了智能网络系统的安全性的一个初步讨论，但缺乏可实现的解决方案或哦给你做思路。我们认为，在 TranSys 的基础上建立可自我解释的模型可以防止深度神经网络做出错误得离谱的决定[90]。同时，引入对抗机器学习或许也是增强智能网络系统安全性的一个思路[91]。同时，最近还有一些研究工作通过增加基于编程语言的盾牌来去组织深度神经网络做出不安全的决定[92]。我们也可以采用类似方案，并应用到智能网络系统中。我们将其留作下一步工作。

可扩展性。网络系统一般来说是一直在运行的，并且高度依赖于输入。例如，对于拥塞控制来说，即使采取相同的当前决策，不同的未来网络流也会导致不同的性能。因此智能网络系统训练上十分困难[93]。当前方法是否能够扩展到更复杂的网络系统中仍然未知。尽管如此，TranSys 或许可以扩展到更深的神经网络，这是因为更深的神经网络和浅层神经网络的表达能力相同[10]，尽管训练难度不同。我们仍需要对 TranSys 在更深的网络系统中的效果进行研究。

插图索引

表格索引

参考文献

[1] Chrome Devtools[EB/OL]. https://developers.google.com/web/tools/chrome-devtools/.

[2] Dash.js[EB/OL]. https://github.com/Dash-Industry-Forum/dash.js/.

[3] Grunt: The javascript task runner[EB/OL]. https://gruntjs.com/.

[4] Tensorflow.js: A javascript library for training and deploying ml models in the browser and on node.js[EB/OL]. https://js.tensorflow.org/.

[5] Abadi M, Barham P, Chen J, et al. Tensorflow: A system for large-scale machine learning[C]//12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). 2016: 265-283.

[6] Adadi A, Berrada M. Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI)[J]. IEEE Access, 2018, 6: 52138-52160.

[7] Aghaei S, Azizi M J, Vayanos P. Learning Optimal and Fair Decision Trees for Non-Discriminative Decision-Making[C]//33rd AAAI Conference on Artificial Intelligence (AAAI 19).

[8] Alizadeh M, Greenberg A, Maltz D A, et al. Data center tcp (dctcp)[C]. ACM SIGCOMM computer communication review, 2011, 41(4): 63-74.

[9] Alizadeh M, Yang S, Sharif M, et al. pfabric: Minimal near-optimal datacenter transport[C]//ACM SIGCOMM Computer Communication Review. ACM, 2013, 43(4): 435-446.

[10] Ba J, Caruana R. Do deep nets really need to be deep?[C]//Advances in neural information processing systems. 2014: 2654-2662.

[11] Bai W, Chen L, Chen K, et al. Information-agnostic flow scheduling for commodity data centers[C]//12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15). 2015: 455-468.

[12] Bastani O, Kim C, Bastani H. Interpretability via model extraction[J]. arXiv preprint arXiv:1706.09773, 2017.

[13] Bastani O, Pu Y, Solar-Lezama A. Verifiable reinforcement learning via policy extraction[C]//Advances in Neural Information Processing Systems. 2018: 2494-2504.

[14] Bau D, Zhou B, Khosla A, et al. Network dissection: Quantifying interpretability of deep visual representations[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017: 6541-6549.

[15] Blockeel H, De Raedt L. Top-down induction of first-order logical decision trees[J]. Artificial intelligence, 1998, 101(1-2): 285-297.

[16] Bosshart P, Daly D, Gibb G, et al. P4: Programming protocol-independent packet processors[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(3): 87-95.

[17] Bosshart P, Gibb G, Kim H S, et al. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN[J]. ACM SIGCOMM Computer Communication Review, 2013, 43(4): 99-110.

[18] Bucilua C, Caruana R, Niculescu-Mizil A. Model compression[C]//Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2006: 535-541.

[19] Carroll A, Heiser G. An Analysis of Power Consumption in a Smartphone[C]//USENIX annual technical conference. 2010, 14: 21-21.

[20] Chen C, Zhang M, Liu Y, et al. Neural attentional rating regression with review-level explanations[C]//Proceedings of the 2018 World Wide Web Conference on World Wide Web. International World Wide Web Conferences Steering Committee, 2018: 1583-1592.

[21] Chen L, Lingys J, Chen K, et al. AuTO: scaling deep reinforcement learning for datacenter-scale automatic traffic optimization[C]//Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. ACM, 2018: 191-205.

[22] Chen M. Minimalrnn: Toward more interpretable and trainable recurrent neural networks[J]. arXiv preprint arXiv:1711.06788, 2017.

[23] Chen W, Wilson J, Tyree S, et al. Compressing neural networks with the hashing trick[C]//International Conference on Machine Learning. 2015: 2285-2294.

[24] Chen Y H, Krishna T, Emer J S, et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks[J]. IEEE Journal of Solid-State Circuits, 2017, 52(1): 127-138.

[25] Cheng Z, Chang X, Zhu L, et al. MMALFM: Explainable recommendation by leveraging reviews and images[J]. ACM Transactions on Information Systems (TOIS), 2019, 37(2): 16.

[26] Raw data – measuring broadband America[EB/OL]. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016.

[27] Dauphin Y N, Bengio Y. Big neural networks waste capacity[J]. arXiv preprint arXiv:1301.3583, 2013.

[28] Deng J, Dong W, Socher R, et al. Imagenet: A large-scale hierarchical image database[C]//2009 IEEE conference on computer vision and pattern recognition. Ieee, 2009: 248-255.

[29] Dobrian F, Sekar V, Awan A, et al. Understanding the impact of video quality on user engagement[C]//ACM SIGCOMM Computer Communication Review. ACM, 2011, 41(4): 362-373.

[30] Ecarlat P. CNN – Do we need to go deeper?[EB/OL]. https://medium.com/finc-engineering/cnn-do-we-need-to-go-deeper-afe1041e263e.

[31] Fang B, Zeng X, Zhang M. NestDNN: Resource-Aware Multi-Tenant On-Device Deep Learning for Continuous Mobile Vision[C]//Proceedings of the 24th Annual International Conference on Mobile Computing and Networking. ACM, 2018: 115-127.

[32] Firestone D, Putnam A, Mundkur S, et al. Azure accelerated networking: SmartNICs in the public cloud[C]//15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18). 2018: 51-66.

[33] Breiman L. Classification and regression trees[M]. Routledge, 2017.

[34] Gawlowicz P, Zubow A. ns3-gym: Extending openai gym for networking research[EB/OL]. https://apps.nsnam.org/app/ns3-gym/.

[35] Gehr T, Mirman M, Drachsler-Cohen D, et al. Ai2: Safety and robustness certification of neural networks with abstract interpretation[C]//2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018: 3-18.

[36] Greenberg A, Hamilton J R, Jain N, et al. VL2: a scalable and flexible data center network[C]//ACM SIGCOMM computer communication review. ACM, 2009, 39(4): 51-62.

[37] Guo W, Mu D, Xu J, et al. Lemna: Explaining deep learning-based security applications[C]//Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2018: 364-379.

[38] Han S, Shen H, Philipose M, et al. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints[C]//Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services. ACM, 2016: 123-136.

[39] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[40] Huang T Y, Johari R, McKeown N, et al. A buffer-based approach to rate adaptation: Evidence from a large video streaming service[J]. ACM SIGCOMM Computer Communication Review, 2015, 44(4): 187-198.

[41] Hussein A, Gaber M M, Elyan E, et al. Imitation learning: A survey of learning methods[J]. ACM Computing Surveys (CSUR), 2017, 50(2): 21.

[42] Jiang J, Sekar V, Zhang H. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive[J]. IEEE/ACM Transactions on Networking (ToN), 2014, 22(1): 326-340.

[43] Konda V R, Tsitsiklis J N. Actor-critic algorithms[C]//Advances in neural information processing systems. 2000: 1008-1014.

[44] Krishnan S, Yang Z, Goldberg K, et al. Learning to optimize join queries with deep reinforcement learning[J]. arXiv preprint arXiv:1808.03196, 2018.

[45] Krishnan S S, Sitaraman R K. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs[J]. IEEE/ACM Transactions on Networking, 2013, 21(6): 2001-2014.

[46] Lane N D, Georgiev P, Qendro L. DeepEar: robust smartphone audio sensing in unconstrained acoustic environments using deep learning[C]//Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing. ACM, 2015: 283-294.

[47] Li H, Kadav A, Durdanovic I, et al. Pruning filters for efficient convnets[J]. arXiv preprint arXiv:1608.08710, 2016.

[48] Liu W, Chawla S, Cieslak D A, et al. A robust decision tree algorithm for imbalanced data sets[C]//Proceedings of the 2010 SIAM International Conference on Data Mining. Society for Industrial and Applied Mathematics, 2010: 766-777.

[49] Lundberg S M, Lee S I. A unified approach to interpreting model predictions[C]//Advances in Neural Information Processing Systems. 2017: 4765-4774.

[50] Lyu D, Yang F, Liu B, et al. SDRL: Interpretable and Data-efficient Deep Reinforcement Learning Leveraging Symbolic Planning[J]. arXiv preprint arXiv:1811.00090, 2018.

[51] MacQueen J. Some methods for classification and analysis of multivariate observations[C]//Proceedings of the fifth Berkeley symposium on mathematical statistics and probability. 1967, 1(14): 281-297.

[52] Mao H, Alizadeh M, Menache I, et al. Resource management with deep reinforcement learning[C]//Proceedings of the 15th ACM Workshop on Hot Topics in Networks. ACM, 2016: 50-56.

[53] Mao H, Netravali R, Alizadeh M. Neural adaptive video streaming with pensieve[C]//Proceedings of the Conference of the ACM Special Interest Group on Data Communication. ACM, 2017: 197-210.

[54] Mao H, Schwarzkopf M, Venkatakrishnan S B, et al. Learning scheduling algorithms for data processing clusters[C]//Accepted by the Conference of the ACM Special Interest Group on Data Communication. ACM, 2019.

[55] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.

[56] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540): 529.

[57] Montazeri B, Li Y, Alizadeh M, et al. Homa: A receiver-driven low-latency transport protocol using network priorities[C]//Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. ACM, 2018: 221-235.

[58] Morawiec D. Sklearn-porter[EB/OL]. https://github.com/nok/sklearn-porter.

[59] Netravali R, Sivaraman A, Das S, et al. Mahimahi: Accurate Record-and-Replay for HTTP[C]//2015 USENIX Annual Technical Conference (USENIX ATC 15). 2015: 417-429.

[60] Netronome White Paper: NFP-4000 Theory of Operation[EB/OL]. https://www.netronome.com/media/documents/WP_NFP4000_TOO.pdf.

[61] Tofino 2 | Barefoot Networks[EB/OL]. https://www.barefootnetworks.com/products/brief-tofino-2/.

[62] Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: Machine learning in Python[J]. Journal of machine learning research, 2011, 12(Oct): 2825-2830.

[63] Ribeiro M T, Singh S, Guestrin C. Why should i trust you?: Explaining the predictions of any classifier[C]//Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, 2016: 1135-1144.

[64] Ribeiro M T, Singh S, Guestrin C. Semantically equivalent adversarial rules for debugging nlp models[C]//Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2018: 856-865.

[65] Riiser H, Vigmostad P, Griwodz C, et al. Commute path bandwidth traces from 3G networks: analysis and applications[C]//Proceedings of the 4th ACM Multimedia Systems Conference. ACM, 2013: 114-118.

[66] Ruffy F, Przystupa M, Beschastnikh I. Iroko: A Framework to Prototype Reinforcement Learning for Data Center Traffic Control[J]. arXiv preprint arXiv:1812.09975, 2018.

[67] Salman S, Streiffer C, Chen H, et al. DeepConf: Automating Data Center Network Topologies Management with Machine Learning[C]//Proceedings of the 2018 Workshop on Network Meets AI & ML. ACM, 2018: 8-14.

[68] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.

[69] Selvaraju R R, Cogswell M, Das A, et al. Grad-cam: Visual explanations from deep networks via gradient-based localization[C]//Proceedings of the IEEE International Conference on Computer Vision. 2017: 618-626.

[70] Shrikumar A, Greenside P, Kundaje A. Learning important features through propagating activation differences[C]//Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017: 3145-3153.

[71] Spiteri K, Urgaonkar R, Sitaraman R K. BOLA: Near-optimal bitrate adaptation for online videos[C]//IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications. IEEE, 2016: 1-9.

[72] Stockhammer T. Dynamic adaptive streaming over HTTP--: standards and design principles[C]//Proceedings of the second annual ACM conference on Multimedia systems. ACM, 2011: 133-144.

[73] Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. MIT press, 2018.

[74] Sutton R S, McAllester D A, Singh S P, et al. Policy gradient methods for reinforcement learning with function approximation[C]//Advances in neural information processing systems. 2000: 1057-1063.

[75] Sze V, Chen Y H, Yang T J, et al. Efficient processing of deep neural networks: A tutorial and survey[J]. Proceedings of the IEEE, 2017, 105(12): 2295-2329.

[76] Valadarsky A, Schapira M, Shahaf D, et al. Learning to route[C]//Proceedings of the 16th ACM Workshop on Hot Topics in Networks. ACM, 2017: 185-191.

[77] Venables W N, Ripley B D. Tree-based methods[M]//Modern applied statistics with S-Plus. Springer, New York, NY, 1999: 303-327.

[78] Verma A, Murali V, Singh R, et al. Programmatically Interpretable Reinforcement Learning[C]//International Conference on Machine Learning. 2018: 5052-5061.

[79] Wu Y, Mansimov E, Grosse R B, et al. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation[C]//Advances in neural information processing systems. 2017: 5279-5288.

[80] Yeo H, Do S, Han D. How will Deep Learning Change Internet Video Delivery?[C]//Proceedings of the 16th ACM Workshop on Hot Topics in Networks. ACM, 2017: 57-64.

[81] Yeo H, Jung Y, Kim J, et al. Neural adaptive content-aware internet video delivery[C]//13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). 2018: 645-661.

[82] Yin X, Jindal A, Sekar V, et al. A control-theoretic approach for dynamic adaptive video streaming over HTTP[C]//ACM SIGCOMM Computer Communication Review. ACM, 2015, 45(4): 325-338.

[83] Zhang Q, Nian Wu Y, Zhu S C. Interpretable convolutional neural networks[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018: 8827-8836.

[84] Zheng Y, Liu Z, You X, et al. Demystifying deep learning in networking[C]//Proceedings of the 2nd Asia-Pacific Workshop on Networking. ACM, 2018: 1-7.

[85] Ziebart B D, Maas A L, Bagnell J A, et al. Maximum entropy inverse reinforcement learning[C]//Aaai. 2008, 8: 1433-1438.

[86] Ghodsi A, Zaharia M, Hindman B, et al. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types[C]//Nsdi. 2011, 11(2011): 24-24.

[87] Goyal P, Agarwal A, Netravali R, et al. ABC: A Simple Explicit Congestion Control Protocol for Wireless Networks[C]//Accepted by the Conference of the ACM Special Interest Group on Data Communication. ACM, 2019.

[88] Hardt M, Price E, Srebro N. Equality of opportunity in supervised learning[C]//Advances in neural information processing systems. 2016: 3315-3323.

[89] Looks M, Herreshoff M, Hutchins D L, et al. Deep learning with dynamic computation graphs[J]. arXiv preprint arXiv:1702.02181, 2017.

[90] Rudin C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead[J]. Nature Machine Intelligence, 2019, 1(5): 206.

[91] Usama M, Qadir J, Al-Fuqaha A, et al. The Adversarial Machine Learning Conundrum: Can The Insecurity of ML Become The Achilles' Heel of Cognitive Networks?[J]. arXiv preprint arXiv:1906.00679, 2019.

[92] Zhu H, Xiong Z, Magill S, et al. An inductive synthesis framework for verifiable reinforcement learning[C]//Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. ACM, 2019: 686-701.

[93] Mao H, Venkatakrishnan S B, Schwarzkopf M, et al. Variance Reduction for Reinforcement Learning in Input-Driven Environments[J]. arXiv preprint arXiv:1807.02264, 2018.

致 谢

衷心感谢我的导师毕军教授。从我 2016 年加入实验室工作，至今已近三年。每每回想起，竟还像是昨天一样。尤其是在我面临留校保研还是出国深造的抉择时，您一次次与我交流，描绘实验室的宏伟蓝图；同时也设身处地为我规划，支持我前往国外研修。在我投稿不顺，论文一次次被拒时，您通过各种方式来鼓励我，并指导我继续修改完善。您奋力为我们营造出自由、宽松的学术环境，并在科研、学习、生活等等各个方面无微不至地给予我们大力支持，现在看来，不知耗费了您多少心血。本文完成之日，正是您下葬的日子。师恩难忘，请您一路走好！

特别感谢王继龙老师、徐明伟老师及王旸旸老师在后续工作中对我的指导。没有您们，我难以将工作继续下去，并完成本篇论文。还要感谢 Clemson 大学的胡宏新老师，麻省理工学院的 Mohammad Alizadeh 老师、北京大学的杨仝老师、中科院计算所的黄群老师对我的指导。您们对我论文、学术等的指导，让我的学术水平有了提高。

还要感谢清华大学的孙晨、郑智隆、于恒、王舒鹤、高凯、徐安民、白家松等师兄，张肖、王海萍、操佳敏等师姐，同小组的王敏虎、陈婧、郭雅宁等同学，以及麻省理工学院的毛宏自师兄，在几年来，从设计、实现上对我的帮助建议。还有许多的老师、师兄师姐、同学对我的工作提出了许多宝贵的建议，在此一并表示感谢。

最后，感谢室友对我的帮助，感谢家人对我的支持，感谢女朋友对我的陪伴。谢谢你们！

谨以此文，献给毕军老师。

声 明

本人郑重声明：所呈交的学位论文，是在导师指下独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用内容外本学位论研究成果不包含任何他人享有著作权的内容。对本论文所涉及研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签 名：[signature] 日 期：2019.6.21

附录 A 外文资料的书面翻译

论文索引：C. Sun, J. Bi, Z. Meng, T. Yang, X. Zhang and H. Hu, "Enabling NFV Elasticity Control With Optimized Flow Migration," in *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2288-2303, Oct. 2018.

NFV 弹性控制中的优化流迁移策略

Chen Sun[1], Jun Bi[1], Zili Meng[1], Tong Yang[2], Xiao Zhang[1], Hongxin Hu[3]

[1]Tsinghua University, [2]Peking University, [3]Clemson University

摘要——网络功能虚拟化（NFV）和软件定义网络（SDN）提供了灵活服务分发和低成本的潜力。基于能够动态创建与销毁网络功能实例的能力，NFV 在如 NF 向外扩容、向内收缩、负载均衡等 NF 管理上提供了很大的弹性。为了实现 NFV 弹性控制，网络流量需要在不同 NF 实例之间重新分配。然而，决定哪一条流适合迁移是高效 NFV 弹性控制中的关键问题之一。在本文中，我们提出一种全新的流控制管理器，OFM 控制器，来获得 NFV 弹性控制下的优化流迁移。我们分析了不同场景下的触发条件与控制目标，并为了回应包含避免队列溢出、迁移成本计算、有效流迁移选择的三大挑战而仔细设计了模块和算法。我们在 NFV 和 SDN 环境上端部署了 OFM 控制器。试验结果表明 OFM 可以高效地支持 NFV 弹性控制场景下的优化流迁移。

索引项——弹性控制，优化流迁移，用户等级协议，网络功能虚拟化。

A.1 引言

最近学术界推出了网络功能虚拟化（NFV）[2]，用基于软件的网络功能（NF）取代传统的专用硬件中间盒，以增强服务交付的灵活性，并降低总体成本。基于动态地对 NF 创建和销毁的能力，NFV 可以支持对 NF 实例的弹性控制，以适应网络流量的频繁性和动态性的本质[3,4]。

NFV 的弹性已经在诸如数据中心[5]和 5G 网络[6,7]的现实世界网络中被广泛利用。例如，5G 网络提供具有高吞吐量，超低延迟或大规模连接的隔离网络片。每个片包含由部署在虚拟机中的虚拟网络功能（VNF）组成的网络服务（NS）。为了适应网络切片中不断增加的流量负载，最近的研究提出通过动态部署携带 VNF 或 NS 实例的新虚拟机（VM）来扩展单个 VNF[8]或整个 NS[9]。扩展后，使用前端物理或虚拟交换机作为负载均衡器，将流分布在一组相同的实例中[5,9]。

此外，NF 通常必须维护已处理流的状态信息[10,11]。为了确保流重新分配后数据包处理的正确性，一些研究工作已提出将流状态与流迁移一起转移[5,12-16]。Split/Merge[14]和 OpenNF[5]依靠集中控制器在 NF 实例和缓冲传入数据包之间传输状态，以实现无损和保持顺序的迁移。另一方面，改进过的 OpenNF[12]和其他工作[13,15]在 NF 实例之间直接执行迁移，以提高 NFV 网络中流迁移的可扩展性和性能。以上研究工作主要集中在为 NF 实例中的流状态的安全迁移设计机制。

然而，选择合适的迁移流量也是 NFV 弹性控制中的重要问题。不经意地选择迁移流程会产生三个主要问题：

- **队列溢出。**从系统的角度来看，流迁移需要目标 NF 中的预分配缓冲区来存储迁移中的流量[12,15]。迁移中流量是指在迁移状态后到达源实例的流量，或者是在相应状态变为可用之前到达目标实例的流量。粗心选择的流可能导致将多个大流迁移到一起，这可能会溢出缓冲区队列空间并导致数据包丢失或服务降级。
- **高迁移代价。**从网络租户的角度来看，NFV 网络应该满足服务水平协议（SLA）。违反某些 SLA 将会受到处罚。但是，流迁移可能会带来额外的处理延迟（数十毫秒[15]）。这对一部分有着很紧的延迟 SLA 的流是不可接受的（例如算法股票交易流程或高性能分布式内存缓存[17]），而对有着较宽松限制的流是可接受的（如 P2P 传输流）。因此，随机选择要迁移的流可能会导致严重的 SLA 违规并显着增加迁移成本。
- **低效迁移。**从网络运营商的角度来看，在没有适当的流量选择机制的情况下实现 NFV 弹性控制可能无法实现控制期望目标。例如，当 NF 实例过载时，选择太少的流量进行迁移可能无法有效缓解热点，而迁移太多流量可能会产生新的热点。

为了解决上述问题，在本文中，我们提出了一种新颖的流量迁移控制器，OFM 控制器，用于优化 NFV 弹性控制中的流量迁移。据我们所知，这是第一个为 NFV 弹性控制执行优化的流量选择控制器。我们分析 NFV 弹性控制情况并仔细设计

OFM 控制器以完全实现控制目标，最小化迁移成本并避免缓冲区队列溢出。我们在本章中做出以下贡献：

- 我们将典型的 NFV 弹性控制情况分类，包括 NF 缩放，NF 负载平衡，NF 故障恢复和 NF 升级。我们详细分析了每种情况的触发条件和流量选择目标，并提出了设计挑战（§A.2）。
- 我们提出了 OFM 控制器的设计，以优化 NFV 弹性控制中的流动偏移。OFM 控制器在运行时收集流统计信息和 NF 加载，并确定需要流迁移的情况。通过有效地建模缓冲要求和迁移延迟（§A.3），OFM 可以选择适当的流来实现控制目标，同时最小化迁移成本并避免缓冲区溢出（§A.4）。
- 我们基于 Floodlight 实现 OFM 控制器并执行广泛的评估。实验结果表明，OFM 可以在 NFV 弹性控制中实现优化的流动迁移，同时确保完全实现控制目标（§A.5）。

本文的其余部分安排如下。第 A.2 分析 NFV 弹性控制的情况。第 A.3 节详细阐述了 OFM 设计。第 A.4 节提供了 OFM 中优化的流迁移计算。我们在第 A.5 节介绍了实现和评估结果。我们在第 A.7 节总结了相关的工作，并结束了本文。

A.2 网络功能动态扩展场景分析

本节首先总结了 NFV 弹性控制需要流量迁移的情况。然后我们分析每种情况的控制目标和约束以及设计挑战，以指导 OFM 的设计。

A.2.1 场景总结

表 A.1 网络功能动态扩展场景分析

场景	何时迁移	迁移原因	迁移到哪里	迁移哪些流
NF 过载	NF 负载 > 峰值负载阈值	避免性能下降	未过载或新创建的实例 (动态扩展)	一部分 (哪些流?)
NF 低载	NF 负载 < 谷值负载阈值	节约资源提高能源利用效率	合并现有实例 (动态收缩)	一部分实例上的所有流 (哪些实例?)
负载均衡	NF 有不均衡的负载	避免潜在过载	在现有实例中	一部分 (哪些流?)
故障恢复	NF 实例出现故障	实现故障恢复	未故障的实例	全部
NF 升级	NF 特性需要升级	执行新的规则	已升级的实例	全部

我们在表 A.1 中列出了 NFV 弹性控制的五种典型情况，并在本节中分析了这些情况。

- **NF 动态扩展:**当 NF 实例的负载超过 NF 处理负载阈值时，会发生这种情况[5,14,18,19]。网络运营商可以在运行时执行 NF 扩展，以通过将一部分流

从过载实例迁移到其他实例或新创建的实例来缓解热点并避免性能下降。但是，过载实例上的流具有不同 SLA 约束和不同流大小。我们应选择适当的流量来缓解热点并且不会产生新的热点，同时导致最小的 SLA 违规并避免缓冲区溢出。

- **NF 动态收缩:**为了节省资源并实现能源效率，当一个或多个 NF 实例负载不足时，通过销毁一部分虚拟机并将这些实例上的所有流迁移到剩余的虚拟机来执行 NF 收缩[5,14,18,19]，这可以节省运营成本（OPEX）。在本文中，我们将节约运营成本与提升“收入收益”等价[20]。但是，流迁移会导致额外的延迟，并且可能违反某些流的 SLA。因此，我们应该选择适当的 NF 实例来销毁，以获得最大的收益和最小的迁移成本。
- **NF 负载均衡:**NF 负载平衡在当前 NF 实例之间重新分配流，以防止潜在的 NF 过载情况。NF 负载平衡不会带来收益，因为它不会关闭虚拟机。但是，流迁移可能会带来额外的转发延迟并导致 SLA 违规处罚。因此，我们应该选择合适的流进行迁移，以平衡负载并最大限度地降低迁移成本。
- **NF 故障恢复:**当一个 NF 实例出现故障时，我们需要通过将故障实例上的所有流重新路由到正常实例或通过创建新实例来从失败中恢复[5]。
- **NF 升级:**为了最大限度地提高安全性，网络提供商可能希望始终使用最新的 NF 软件处理流量[5]。NFV 提供动态启动更新的 NF 实例的功能。我们需要将全部流和状态迁移到更新的实例。

A.2.2 扩展目标

从上面的分析中，我们观察到包括 NF 扩展，收缩和负载平衡在内的情况需要仔细选择流来实现控制目标，同时最小化迁移成本并避免缓冲区队列溢出。因此，我们接下来分析在应对每种情况时的详细流量选择目标，如图 A.1 所示。

- **NF 扩展:**当 NF 过载时，NF 缩小必须以执行以避免数据包丢失或性能下降。运营商希望在不创建新热点的情况下实现快速负载缓解。此外，需要最小的迁移成本，并应避免缓冲区溢出。
- **NF 收缩:**将多个实例合并为更少的实例并销毁免费虚拟机可以提高能效并带来收入收益，我们希望最小化剩余实例的数量。但是，不同实例上的流具有不同的 SLA 约束，我们希望同时最小化迁移成本。因此，我们需要比较 SLA 惩罚，以便将每个实例上的流迁移到销毁虚拟机所带来的收

入收益，并找到最佳迁移计划。此外，将多个实例合并到一个实例上需要安全地扩展，而不会产生新的热点。最后，在迁移期间应避免缓冲区溢出。

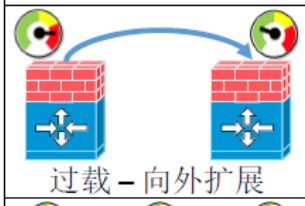
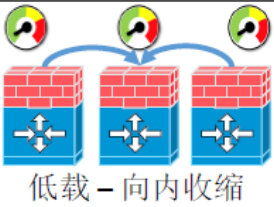
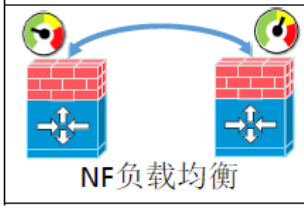
场景	流选择目标
	- 快速缓解热点 - 避免产生新热点 - 最小化迁移成本 - 避免队列溢出
	- 最大化收入收益 - 最小化迁移成本 - 避免产生新热点 - 合理计算时间 - 避免队列溢出
	- 均衡的NF负载 - 最小化迁移成本 - 快速收敛到均衡 - 避免队列溢出

图 A.1 不同场景下的流迁移目标

- **NF 负载均衡:**负载平衡可以平衡 NF 实例之间的负载并防止潜在的 NF 过载情况。但是，NF 负载平衡既不是强制性的（如扩展），也不是会直接产生收益（如收缩）。因此，为了最小化流迁移成本，我们应该只使用在迁移期间不会违反的松散 SLA 重新分配流。因此，只能重新分配有限的一组流，这可能不会导致完全平衡的最终负载。但是，我们可以在一定程度上改善负载不平衡情况而不需要任何成本。

在 E2[17]中提出的用于 NFV 弹性控制的草人解决方案采用了迁移避免的策略。现有流仍由先前分配的 NF 实例处理，而仅差分处理新流。这样，NFV 弹性控制不会发生流动迁移。例如，对于 NF 扩展，我们可以简单地实例化一个新的 NF 实例并将新流重定向到它。虽然迁移避免策略不会引入迁移惩罚，但它仍可能在运行时导致惩罚。如果 NF 过载，我们应该快速从实例迁移流，以避免性能下降和 SLA 违规。我们在第 A.6 节中详细分析了迁移规避策略。

A.2.3 研究挑战

为了实现上述流量选择目标，我们设计了用于 NFV 弹性控制的 OFM 控制器。我们在 OFM 的设计中遇到了三个主要挑战。

避免缓冲区队列溢出：安全弹性控制需要在迁移期间缓冲目标实例中的飞行中流量[12,15]。但是，缓冲空间不是无限的。我们观察到不同流量的迁移会导致不同的飞行中流量。因此，在选择要迁移的流时必须小心，以避免缓冲区溢出。为此，OFM 在不侵入 NF 逻辑的前提下动态测量 NF 实例上的流的大小，并为每个流的迁移建模缓冲区空间要求进行建模（§A.3.B）。

计算迁移成本：流迁移可能会带来额外的转发延迟，违反 SLA 约束并导致惩罚。但是，惩罚取决于 SLA 违反的程度，即超过 SLA 约束的延迟时间。因此，OFM 面临着精确估计迁移延迟的挑战，迁移延迟可能会随迁移流量的数量而显着变化[5,15]。为此，通过对现实世界 NF 的实验，OFM 根据要迁移的流的数量构建流迁移延迟模型，并使用它来计算迁移成本（§A.3.C）。

迁移时高效地流选择：如在§A.2.B 中分析的那样，不同的控制情况具有不同的控制目标。因此，我们面临着为这三种情况设计优化流量选择机制的挑战。但是，大量的参数包括 NF 负载，流量大小（在[21,22]中定义的大流或小流），不同大小的流的迁移延迟，虚拟机收益和缓冲成本应该考虑到找到针对每种情况的优化迁移计划，使设计算法以优化流量选择变得具有挑战性。此外，计算可能消耗大量时间，这对于需要快速热点缓解的 NF 缩小等情况可能是不可接受的。为了解决上述挑战，OFM 精心设计了一个统一，最优但复杂的算法，可以同时处理这三种情况，并同时考虑到上述所有参数，分别针对三种情况提出三种快速但次优的算法（§A.6）。

A.3 OFM 设计

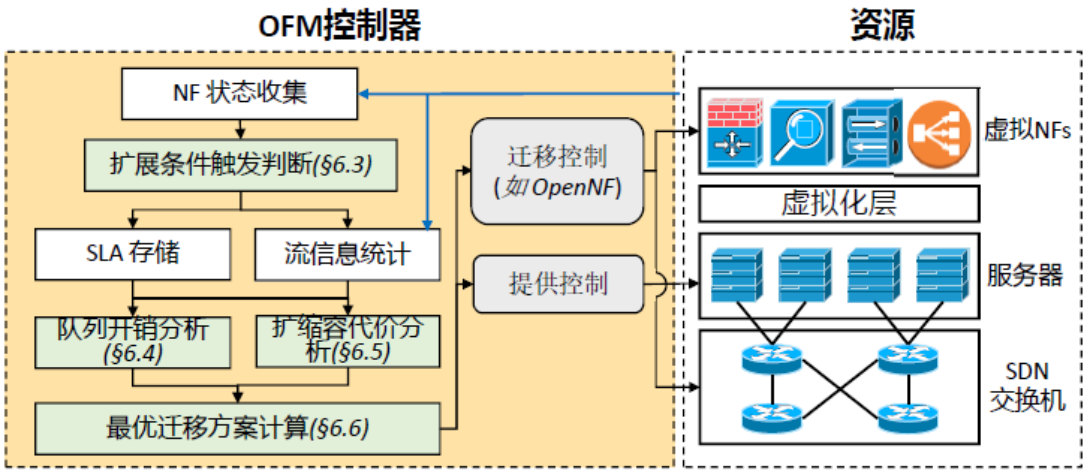


图 A.2 OFM 控制器组成部分与工作流

为了解决上述挑战，我们设计了 OFM 控制器，以在 NFV 弹性控制情况下实现优化的流量迁移。OFM 控制器的组件和工作流程，如图 A.2 所示。OFM 控制器监控每个 NF 实例的状态，并检测流量过载，欠载和不平衡的条件。同时，OFM 控制器收集每个 NF 上的流量统计信息以供进一步选择（§A.3.A）。一旦检测到条件满足，基于不同的流 SLA 约束和动态收集的流统计信息，OFM 控制器首先执行缓冲成本分析（§A.3.B）和迁移成本分析（§A.3.C）。将分析结果输入最优迁移计算模块（§A.6）以创建优化的迁移计划。最后，OFM 供应控制和迁移控制模块将与底层资源交互，以与引入的相同方式执行流迁移[5,12,14,15]等。

但是，一个自然的问题是根据现有的流统计信息计算未来的迁移的优化计划的实用性。实际上，正如[23]中所提到的，我们应该能够在最后 1 秒内使用基于历史流量模式的路由来进行有效的流量调度。此外，如§A.7 所示，OFM 可以在 1 秒内完成所有情况的统计和计算，这证明了 OFM 的及时性。接下来我们详细介绍 OFM 的各个模块。

A.3.1 NF 和流的状态收集与条件触发

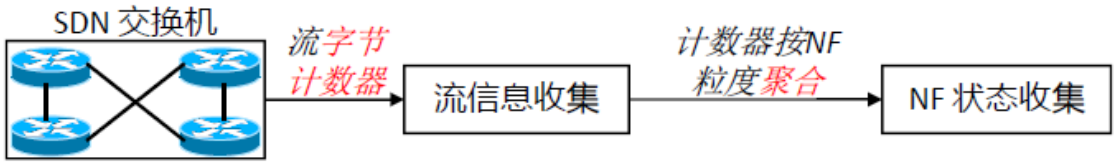


图 A.3 OFM 中的状态分析

OFM 控制器需要收集 NF 处理负载，即，吞吐量，弹性控制条件检测，以及流量选择的流量大小。获取这些统计信息的一种简单方法是修改 NF 逻辑以维护流级别数据包计数器。但是，这样做会侵入 NF 逻辑并增加统计信息收集和与控制器通信的 NF 开发负担。为了以轻量级的方式精确地收集上述统计数据，我们利用了 OpenFlow 的流表项计数器[24]。物理或虚拟 OpenFlow 交换机广泛用于连接 NFV 网络中同一服务器中的物理服务器或虚拟机[5,25-27]。OpenFlow 交换机为每个流表条目维护一个字节计数器，而控制器在运行时查询来自交换机的计数器。但是，OpenFlow 流表中的流条目通常是聚合的[28]。直接查询计数器不能提供流级字节计数器。因此，我们使用 OpenFlow 的多阶段流表[24]，将与 NF 连接的边界交换机的第一个流表分配为计数表，并向其发出细粒度规则以维护流级计数器。计数器表中每个条目的操作是直接将数据包发送到下一个流表。如图 A.3 所示，我们定期从计数器表中查询流量计数器，并通过将计数器差除以查询间隔来计算

流量大小。由于 OFM 控制器可以确认每个流的目标 NF，它会根据目标 NF 对流进行分组，并将针对相同 NF 的流的大小相加，以获得 NF 的实时吞吐量。

假设有n个同类的 NF 实例，如防火墙，在 NFV 网络中正在运。OFM 控制器定期从数据平面查询流统计信息并计算实例$j \in [1,n]$的负载l_j。为条件触发，我们定义th_{top}单个 NF 实例的峰值负载阈值，th_{bottom}为谷值负载阈值。我们用 NF 负载的方差$var(l_1,\cdots,l_n)$来量化负载不均衡程。我们定义最大允许的负载方差为th_{var}。我们定义 NFV 弹性扩缩容的条件为：

- 过载：$l_j \geq th_{top,j}, \forall j \in [1,n]$
- 低载：$l_j \leq th_{bottom,j}, \forall j \in [1,n]$
- 不均衡负载：$var(l_1,\cdots,l_n) \geq th_{var}$

基于上述规则，可以检测 NFV 弹性控制条件，这将触发最佳流量迁移计算以分别处理对应的情况。

A.3.2 队列开销分析

在迁移期间，需要缓冲飞行中的流量，直到状态安装结束。然后，将迁移中的流量刷新到目标 NF 实例进行处理。OFM 控制器采用在[12]中采用的分布式缓冲机制，并缓存目标实例中的迁移中流量。我们的目标是通过以下方式估算迁移中的流量来避免缓冲区溢出。

假设流k的速率为$size_k$需要被迁，流k的迁移时间记为$la_{migration,k}$。在流迁移期间，此流的所有正在迁移的数据包都在目标实例处进行缓冲。因此，所需的总缓冲数据包大小可以建模为：

$$buffer_k = size_k \times la_{migration,k} \quad (A\text{-}1)$$

通过这种方式，我们可以计算迁移每个流的缓冲队列要求，并选择适当的流组成的集合以避免目标实例中的缓冲区溢出。流程迁移时间的估计将在本节后面介绍。

A.3.3 扩缩容代价分析

由于流量迁移引起的额外延迟，NFV 弹性控制可能会破坏流量 SLA[20]并产生惩罚[29]。此外，对于 NF 扩展，关闭负载较低的虚拟机可以带来收益并改善迁移成本。接下来，我们将详细介绍 SLA 违规处罚和收入收益估算。

1）服务等级违背惩罚估计：云服务中与延迟相关的 SLA 规定了特定请求种类的最大处理延迟[29]。同样，我们在 NFV 中定义与延迟相关的 SLA：它调节 NFV

网络处理的每个流的最大延迟，其中 NFV 通过 NF 提供数据包处理服务，包括防火墙，IDS，VPN，负载平衡等[18,19]。

给定一个带有m流量的实例j，给定流量k，让la_k为流量k的延迟，让 LAk 为流量k的 SLA 延迟。显然，la_k应该不大于LA_k。在没有流迁移时，NF 实例j上流k的总延迟等于 NF 处理延迟，也就是说$la_k = la_{processing,j}, \forall k \in [1,m]$。但是，流迁移可能会引入额外的延迟开销。因此，为了在迁移期间满足与延迟相关的 SLA，迁移延迟应满足：

$$la_{migration} \leq LA_k - la_{processing,j}, \forall k \in [1,m] \qquad (A\text{-}2)$$

在流程迁移期间，上述不平等可能会被破坏并导致惩罚。流k中没有被及时处理的流量正是缓冲的迁移中流量，也就是$buffer_k$。根据[28]，我们可以将 SLA 违规惩罚建模为线性函数。我们将惩罚率表示为β，将流量迁移的延迟时间表示为 DT。我们有：

$$Penalty = \alpha + \beta \times buffer \times DT \qquad (A\text{-}3)$$

但是，对于流k，如果未违反其延迟 SLA，则将延迟时间设置为零，并且惩罚应该为零。否则，延迟时间是 SLA 约束上超过的延迟差。因此，我们有：

$$DT_k = max\left(0,\ la_{migration,k} + la_{processing,k} - LA_k\right) \qquad (A\text{-}4)$$

流k的迁移代价可建模为：

$$Penalty_k = \begin{cases} \alpha + \beta \times buffer_k \times DT_k, DT_k > 0 \\ 0, DT_k = 0 \end{cases} \qquad (A\text{-}5)$$

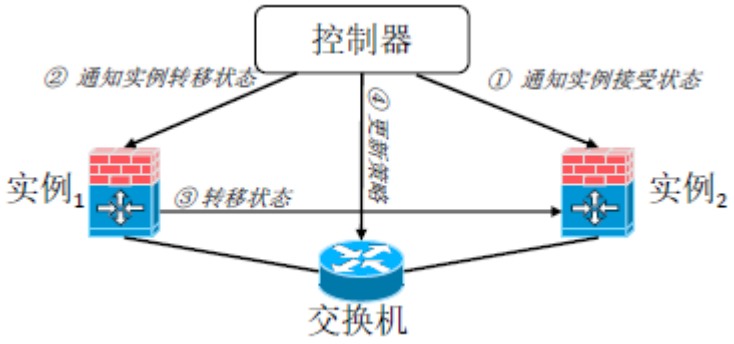


图 A.4 [12]中的四步迁移工作流。交换机可以是物理的也可以是虚拟的。

接下来，我们需要估计流的迁移时间以计算 SLA 成本代价。图 A.4 展示了[12]中的四步迁移，四部分主要时间为：

- t_1：控制器通知目标实例接受状态的时间。

- t_2：控制器通知源实例传输状态的时间。
- ts_k：流k的状态传输时间。
- tu：流规则更新时间。

流k的总迁移时间可表示为：

$$la_{migration,k} = t_1 + t_2 + ts_k + tu \quad (A\text{-}6)$$

其中，t_1，t_2和tu与要迁移哪条流无关。我们可以在 NFV 网络中轻松测量它们并将它们视为常数。但是，如[12]所示，无论流量大小如何，状态传输时间都取决于要迁移的流数。我们在第 A.7 节中的评估表明了流的状态转移时间（ts_k）与要迁移的流的总数（f_n）之间的线性关系。我们将他们的关系描述为：

$$ts_k = \gamma + \eta \times f_n \quad (A\text{-}7)$$

γ和η是两个常量，可能因不同的 NF 类型而异。通过这种方式，我们可以估计选定的流的迁移时间并计算惩罚。迁移f_n条流时，对于单个流k，延迟时间，缓冲要求和迁移成本建模为：

$$DT_k = (t_1 + t_2 + tu + \gamma + \eta \times f_n) + la_{processing,k} - LA_k \quad (A\text{-}8)$$

$$buffer_k = size_k \times (t_1 + t_2 + tu + \gamma + \eta \times f_n) \quad (A\text{-}9)$$

$$cost_k = Penalty_k = \begin{cases} \alpha + \beta \times buffer_k \times DT_k, DT_k > 0 \\ 0, DT_k = 0 \end{cases} \quad (A\text{-}10)$$

2) 收益估计：对于网络功能收缩场景，关闭虚拟机可以带来收益并抵消迁移成本。我们将虚拟机j的价格（即每个时间段的虚拟机的成本[29]）表示为$PriVM_j$。此外，我们需要估计虚拟机扩展所节省的虚拟机运行时间。假设我们总是在低载时关闭虚拟机。则网络功能的平均关闭时间是虚拟机负载低于th_{bottom}的时间。因此，我们收集历史数据，并计算虚拟机低载时的平均时间间隔$TINT_{avg}$，并将其作为估计的节省时间。因此，我们将关闭虚拟机j的收益和总迁移成本建模为：

$$benefit_j = PriVM_j \times TINT_{avg}, \forall j \in [1, m] \quad (A\text{-}11)$$

$$cost_j = \sum_{k=1}^{n} Penalty_k - benefit_j \quad (A\text{-}12)$$

A.4 弹性控制流量迁移方案设计

基于以上建模和分析，在本节中，我们提出了 OFM 控制器算法，以实现优化的 NFV 弹性控制。正如第 A.2 节所述，三种 NFV 弹性控制场景各有独特的流量选择目标。在算法设计之前，我们首先回顾并详细分析流量选择目标。

首先，当一个或多个网络功能实例过载时，OFM 控制器通过将一些流从过载实例迁移到其他实例或新创建的实例来执行网络功能扩展。一种简单的解决方案是迁移走一半流量，以有效缓解热点。但是，过载实例上的流可能具有严格的延时 SLA，迁移这些流中的一半可能会导致严重的处罚。实际上，网络功能扩展的基本控制目标是迁走一些流，以减少网络功能负载至低于峰值阈值以下。为实现此控制目标，我们引入峰值安全阈值th_{safe}，作为网络功能扩展后实例的峰值负载。例如，假设总容量的峰值阈值th_{top}=80%，而th_{safe}=60%。假设有一个过载（80%）实例。我们可以简单地确保迁移 20%的流量，实现有效的过载缓解，而不是迁移一半（40%）负载。实际 NFV 系统中，阈值可以由网络管理员基于网络流量统计来动态配置。

其次，当$N_{us} \geq 1$个网络功能实例低载时，OFM 控制器会通过将多个实例合并到一个实例，并关闭空闲虚拟机来执行网络功能收缩。在实例合并期间，我们追求最大的收益，并通过确保剩余实例的负载低于th_{safe}来避免创造新的热点。

最后，我们定期执行网络功能负载均衡，以防止潜在的网络功能过载情况。最优迁移方案应确保网络功能负载的方差低于峰值方差阈值th_{var}，同时带来最小的迁移处罚。

基于以上分析，下面我们首先提出基于整数线性规划（ILP）的统一最优流迁移方案算法。然而，我们观察到 ILP 无法在有限的时间内快速解决。因此，我们分别利用三种场景的特点来放松约束，加速计算。对于每种场景，我们首先介绍最优的流量选择算法，然后介绍相应的启发式算法，以保证 OFM 的时效性。最后，由于我们针对不同的场景使用不同的算法，我们提出了一种协调机制，来应对多种场景同时出现的情况。

A.4.1 通用方案设计

首先，我们提出了一种算法，可以为所有 NFV 弹性控制场景产生最优的流量迁移方案。设N_s为网络功能实例的数量，N_d为迁移后的实例数。在网络功能扩展期间，在最坏的情况下，当所有N_s个网络功能实例同时过载时，将创建最多N_s个新网络功能实例。在网络功能扩展期间，至少 1 个网络功能实例将保留。因此，

$1 \le N_d \le 2N_s$。我们使用$x_{fsd} \in \{0,1\}$指示流量f是否从源实例s迁移到目标实例d。如果$s = d$，则认为流不会迁移，并不带来任何处罚。简单起见，我们假设回收一个虚拟机可带来$benefit$的收益。假设实例s上有m_s条流。求解x的 ILP 公式是：

$$\min(Penalty - Benefit), where \tag{A-13}$$

$$Penalty = \sum_{s}^{N_s} \sum_{d=1, d \neq s}^{N_d} \left(\sum_{f=1}^{m_s} x_{fsd} \times Penalty_{fsd} \right) \tag{A-14}$$

$$Benefit = \left(N_s - \sum_{d=1}^{N_d} sign\left(\sum_{f,s} x_{fsd} \right) \right) \times benefit \tag{A-15}$$

s.t.

(1) $x_{fsd} \in \{0,1\}, \forall s \in [1, N_s], d \in [1, N_d], f \in [1, m_s]$

(2) $\sum_{d=1}^{N_d} x_{fsd} = 1, \forall\, s \in [1, N_s], f \in [1, m_s]$

(3) $\sum_{s=1}^{N_s} \sum_{f=1}^{m_s} x_{fsd} \times buffer_f \le Buffer_d, \forall d \in [1, N_d]$

(4) $load_s + \sum_{s=1}^{N_s} \sum_{f=1}^{m_s} x_{fid} \times size_f - \sum_{j=1}^{N_d} \sum_{f=1}^{m_s} x_{fsj} \times size_f \le th_{safe}, \forall s \in [1, N_s]$

(5)* $var\left(load_s + \sum_{i=1}^{N_s} \sum_{f=1}^{m_s} x_{fid} \times size_f - \sum_{j=1}^{N_d} \sum_{f=1}^{m_s} x_{fsj} \times size_f \right) \le th_{var}$

其中$size_f$表示流f的大小，$buffer_f$表示迁移流f所需的缓冲区，$Buffer_d$表示目标实例d中的缓冲区大小，$load_s$ 表示当前实例d的负载。目标函数（公式 A-13）最小化总迁移成本。其中，处罚（公式 A-14）来自迁移期间的 SLA 违规处罚，收益（公式 A-15）来自虚拟机实例的销毁。我们认为如果虚拟机上的所有流都被迁移掉，且没有流迁移到它，则虚拟机被关闭。约束（1）规定流f要么迁移要么不迁移，而约束（2）确保流只迁移一次，可迁移到任何实例，包括它所属的实例。约束（3）避免了目标实例中的缓冲区溢出。约束（4）确保不会创建新的热点。约束（5）* 通过约束网络功能负载的方差在阈值th_{var}以下来均衡网络功能负载。

但是，使用约束（1）到（4），算法可以生成多余一个可行的解决方案，因为我们可以动态地创建或破坏 NF 实例以安全地容纳所有流。同时，约束（5）试图限制迁移后 NF 实例负载的方差。在存在两个 NF 实例的极端情况下，每个实例仅携带一个流，并且两个实例的负载分别为 50%和 10%。两个实例的负载无法平衡，使得 ILP 公式无解。因此，我们通过将约束（1）到（4）分配为高目标值并

将低目标值分配给约束（5）来利用目标规划[30]技术。这确保了如果约束（5）使问题无法解决，仍然可以生成满足约束（1）到（4）的可行解。

上述 ILP 公式可以同时处理这三种情况并产生最佳迁移计划。然而，由于两个主要原因，ILP 问题无法快速解决。首先，我们从（公式 A-5）观察到，$Penalty_{fsd}$是一个分段函数，由于DT_{fsd}不连续，使得 ILP 公式在短时间内无法解决（如几毫秒[30]）。其次，我们为每个流分配一个参数x_{fsd}，以指示流f是否从实例 s 迁移到d。参数的数量与 NF 实例的数量（数百[32]）和流的数量（数百万[33]）成正比。大量参数使得快速解 ILP 变得困难。但是，根据 § A.2 中 NF 扩展和负载平衡的控制目标，需要进行高效计算以快速生成迁移计划。为了应对这一挑战，我们利用分别在三种情况下放松约束的机会，并针对每种情况提出快速算法。我们将在本节的其余部分介绍三种快速算法的设计。

A. 4. 2 动态扩展方案设计

1) 动态扩展的最优求解：当一个或多个 NF 实例被重载时，OFM 控制器通过将一些流从重载实例迁移到其他实例或新创建的实例来扩展 NF 实例。如在 § A.2 部分所分析，NF 缩小需要快速热点缓解，而不会产生新的热点或导致缓冲区溢出。我们可以识别重载的实例和将它们视为要迁移的流的源实例。因此，源实例集仅包括过载实例，这会显着减少参数数量。此外，NF 扩展不会在迁移后追求平衡负载。放宽约束也有助于加速 ILP 的解决。我们使用N_{os}来表示重载的 NF 实例的数量。我们有$N_s \le N_d \le N_s + N_{os}$。基于以上观察，我们修改 NF 扩展的目标函数如下。

$$\min(Penalty - Benefit), \qquad where \tag{A-16}$$

$$Penalty = \sum_{s}^{N_s} \sum_{d=1, d\neq s}^{N_d} \left(\sum_{f=1}^{m_s} x_{fsd} \times Penalty_{fsd} \right) \tag{A-17}$$

$$Benefit = \left(N_s - \sum_{d=1}^{N_d} sign\left(\sum_{f,s} x_{fsd} \right) \right) \times benefit \tag{A-18}$$

我们在先前的 ILP 公式中采用约束（1）到（4）作为约束。

2) 动态扩展的快速启发式算法：但是，上面的目标函数仍然是分段的，使得 ILP 在几毫秒的有限时间内无法解决。根据在 § A.2 中缩小 NF 的控制目标，需要有效的计算来快速缓解热点。因此，我们提出了一个三步启发式来加速计算。我们首先选择要从重载实例迁移出来的流，以确保有效缓解过载情况。接下来，

我们计算是否可以将选定的流放置到其他当前部署的实例。如果当前实例不能保存与th_{safe}约束相关的所有流，则我们将尽可能多的流放入当前实例。最后，我们部署新的 NF 实例以容纳无法放在当前实例上的剩余流。直觉是避免启动新的 NF 实例以减少迁移惩罚。接下来我们将详细介绍这三个步骤。

步骤 1：选择流。我们首先在每个重载实例上选择一组流，以最小的迁移成本来缓解热点。由于我们无法预先确认要迁移的流的总数，因此我们假设每个流都是单独迁移的，并且消耗单流迁移时间（Single Flow Migration Time，简称 SFMT）。可以针对不同的 NF 类型测量和计算 SFMT，这将在 § A.7 中介绍。对于每个重载的实例s，我们执行以下简单的 ILP 算法。

$$\min \sum_{f=1}^{m_s} x_{fs} \times Penalty_f, \qquad where \tag{A-19}$$

$$Penalty = size_f \times \left(la_{processing,f} + SFMT - LA_f\right) \tag{A-20}$$

s.t.

(1) $x_{fs} \in \{0,1\}, for\ s \in [1, N_{os}], f \in [1, m_s]$

(2) $load_s - th_{safe} \le \sum_{f=1}^{m_s} x_{fs} \times size_f \le load_s/2$

约束（1）规定流迁移或不迁移。约束（2）确保选择足够的流量以有效减轻热点。根据我们在 § A.7 中的评估，上述 ILP 可以在几毫秒内快速解决。

步骤 2： 放置流到现有实例上。 接下来，我们将先前选择的流放置到未过载的当前实例。在放置期间，我们尝试在当前实例上放置尽可能多的流，以便最小化要部署的新实例的数量。在此过程中，约束是避免新的热点创建和缓冲区溢出。假设将m_f流重新分配到未过载的N_d当前实例上。我们为此步骤提供 ILP 如下。

$$max \sum_{d=1}^{N_d} \sum_{f=1}^{m_f} x_{fd} \times size_f \tag{A-21}$$

s.t.

(1) $x_{fd} \in \{0,1\}, for\ d \in [1, N_d], f \in [1, m_f]$

(2) $\sum_{d=1}^{N_d} x_{fd} \le 1, \forall f \in [1, m_f]$

(3) $load_d + \sum_{d=1}^{N_d} \sum_{f=1}^{m_f} x_{fid} \times size_f \le th_{safe}, \forall d \in [1, N_d]$

(4) $\sum_{f=1}^{m_f} x_{fd} \times buffer_f \leq Buffer_d, \forall d \in [1, N_d]$

我们将在§A.5 中证明上述公式也可以在短时间内解决。我们检查解决方案x以检查是否所有流都放在当前实例上。如果$\sum_{d=1}^{N_d} x_{fd} = 1, \forall f \in [1, m_f]$，则当前实例可以容纳所有流并且算法结束。否则，我们为剩余的流启动新实例。

步骤 3：部署新实例。最后，我们部署新实例以适应剩余的流量。我们将剩余流量的数量表示为m_{rf}。此步骤的目的是最小化保存所有剩余流的新实例的数量，以便最小化惩罚。迁移后新实例的负载应低于th_{safe}，应避免缓冲区溢出。解决方案是迭代地创建新实例以承载最大可能的流量，直到所有流都打包到 NF 实例中。对于每次迭代，我们将流程打包过程制定为以下 ILP 问题。

$$max \sum_{f=1}^{m_{rf}} x_f \times size_f \qquad (A\text{-}22)$$

s.t.

(1) $x_f \in \{0,1\}, \forall f \in [1, m_{rf}]$

(2) $\sum_f^m x_f \times size_f \leq th_{safe}$

(3) $\sum_f^m x_f \times buffer_f \leq buffer$

在每次迭代之后，从剩余流集中移除所选流，直到移除所有流。我们将在§A.5 中证明上述 ILP 问题可以快速解决。

A. 4. 3 动态收缩方案设计

1) 动态收缩的最优求解：OFM 控制器将通过将欠载实例上的流迁移到其余实例并关闭空闲虚拟机 来执行 NF 收缩。OFM 控制器应 ILP 算法以最小化迁移成本。假设有N_{us}欠载的 NF 实例。实例s带有m_s流量。x_{fsd}表示流f是否从实例s迁移到实d。解决x的 ILP 公式旨在最大化收益（公式 A-13）。剩余实例数N_d，满足$0 < N_d \leq N_s$。为了简洁，我们省略了重复的 ILP 公式。

2) 动态收缩的快速启发式算法：但是，上述公式的目标函数仍然是分段的，因为只有当一个实例上的所有流都被迁移出去时，我们才能破坏实例并获得收益。为了加速计算，我们不再将每个流单独迁移到其他实例。相反，我们通过将一个欠载实例上的全部流迁移到另一个实例并销毁自由实例来合并当前实例。为了实现这一目标，我们提出了一种用于 NF 缩放的两步启发式算法。

步骤 1：正收益实例判断。首先，我们需要确定 NF 实例，其破坏带来的收益比其所有流量的迁移成本更高。我们过滤满足$benefit - penalty > 0$ 的实例，并将它们命名为候选实例。假设有N_{cs}这样的实例。

步骤 2：最优实例合并方案。接下来，我们将候选实例合并到与负载约束th_{safe}和缓冲区大小约束$buffer$相关的其他实例。我们使用$x_{sd} \in \{0,1\}$来表示实例s是否合并到实例d上。我们将问题建模为：

$$\max \sum_{s=1}^{N_s} \sum_{d=1}^{N_d} x_{sd} \times (benefit_s - Penalty_s) \qquad (A\text{-}23)$$

s.t.

(1) $x_{sd} \in \{0,1\}, \forall s \in [1, N_{cs}], d \in [1, N_d]$

(2) $\sum_d^N x_{sd} = 1, \forall s \in [1, N_{cs}]$

(3) $load_d + \sum_s^N x_{sd} \times size_s \leq th_{safe}$

(4) $\sum_s^N x_{sd} \times buffer_s < buffer, \forall d \in [1, N_d]$

通过求解上述 ILP 公式，我们可以在可接受的时间内计算 NF 缩放的优化流量选择。我们在 § A.5 中评估算法。

A. 4. 4 负载均衡方案设计

1) 负载均衡的最优求解：尽管 NF 负载平衡可以防止潜在的 NF 过载情况，但它既不是强制性的（如 NF 缩小以缓解热点），也不会立即得到回报（如 NF 扩展带来收益效益）。因此，我们将 NF 实例上的流量迁移到负载较重，即大于平均负载的情况下，在没有发生 SLA 违约的情况下，迁移到具有较轻负载的 NF 实例，以获取零迁移成本。请注意，从一个实例迁移的流可能会被放置到不同的 NF 上。OFM 控制器面临的挑战是避免产生热点并实现相对平衡的负载。一个简单的解决方案是将流分成几个相同大小的组，并根据除法将所有流重新分配到所有实例，以便最小化 NF 实例的负载方差。我们使用x_{fsd}作为流量f是否从实例s迁移到实例d的指标。这里我们只迁移流量f，其迁移惩罚为零。我们为此解决方案提供如下 ILP 公式。

$$\min var\left(load_d + \sum_{s=1}^{N_s} \sum_{f=1}^{m_{fs}} x_{fsd} \times size_f - \sum_{f=1}^{m_{fd}} x_{fsd} \times size_f \right) \qquad (A\text{-}24)$$

s.t.

(1) $x_{fsd} \in \{0,1\}, \forall s \in [1, N_s], d \in [1, N_d], f \in [1, m_f]$

(2) $\sum_{d=1}^{N_d} x_{fsd} \leq 1, \forall s \in [1, N_s], f \in [1, m_f]$

(3) $\sum_{s=1}^{N_s} \sum_{f=1}^{m_s} x_{fsd} \times buffer_f \leq Buffer_d, \forall d \in [1, N_d]$

2) 负载均衡的快速启发式算法：以上全局流重新分配可能导致大量流的迁移，这可能对正常的数据包处理带来负面影响。此外，分段的目标函数使得问题难以在有限的时间内得到解决。因此，我们的主要想法是从负载高于平均负载l_{avg}加上一个标准差l_{stdev}的 NF 实例中获取流，并将所选流重定位到负载低于$l_{avg} - l_{stdev}$的实例。我们设计了一个三步启发式算法，伪代码在 Algorithm 1 中呈现。

步骤 1：实例分类。我们计算 NF 实例的平均负载l_{avg}，并将负载大于$l_{avg} + l_{stdev}$（过载实例）的 NF 实例放入$NFList_{heavy}$，负载低于$l_{avg} - l_{stdev}$（低载实例）放入到$NFList_{light}$。

步骤 2：流选择。对于每个负载很重的实例 s，我们将$l_{avg} + l_{stdev}$以上的额外 NF 负载计算为l_{extras}。我们在实例 s 上选择那些在迁移期间不会违反 SLA 的流。考虑到从一个实例迁移的流可能被放置在多个其他实例上，我们假设每个流都是单独迁移并需要 SFMT 的时间。我们将这些流存储到$FlowList$中，并使用流大小的降序对流进行排序。然后我们逐个选择流进行迁移，直到这一实例在再加上l_{extras}时会变得过载。这是为了快速减少过载实例的负载，因为网络流量可能会有很大差异，并且我们需要快速负载平衡以避免潜在的热点。迁移大流量会减少迁移流的总数并快速达到均衡。

步骤 3：目标 NF 选择。在这一步中，我们将步骤 2 中所有过载实例的选定流合并到最终的$FlowList$中，并将它们拆分到低载实例中以实现平衡负载。我们使用背包算法在$FlowList$中使用选定的流填充每个轻载实例的负载直到其负载达到l_{avg}。我们使用l_{avg}而不是$l_{avg} - l_{stdev}$作为峰值阈值的原因是我们希望在低载实例上容纳尽可能大流量。但是，这时候$FlowList$中的某些流可能仍未分配给任何目标实例。这些流被放回原始的 NF 实例。

基于我们在第 A.5 部分中的评估，上述三步算法可以快速生成迁移计划，以在 NF 实例之间实现相对平衡的负载而不会产生 SLA 违约惩罚。

Algorithm 1: Heuristic Algorithm for NF Load Balancing

Input: Flow Parameters: $size$, $SFMT$, LA, $la_{processing}$
Input: NF Parameters: $load$, l_{avg}, l_{stdev}, $Buffer$.
Output: Flows to Migrate, their sources, and Their Targets: $migration_plan[f, s, d]$.

$NFList_{heavy} = [\,]$, $NFList_{light} = [\,]$, $FlowList = [\,]$, $migration_{plan} = [\,]$;
// Step 1: Instance Classification
foreach $s \in [0, N_s]$ **do**
  if $load_s > l_{avg} + l_{stdev}$ **then**
    $NFList_{heavy}.append(s)$;
  if $load_s < l_{avg} - l_{stdev}$ **then**
    $NFList_{light}.append(s)$;
// Step 2: Flow selection
foreach $s \in NFList_{heavy}$ **do**
  $l_{extra,s} = load_s - (l_{avg} + lstdev)$;
  $totalSize = 0$;
  // Sort flows in according to flow sizes in decending order;
  foreach $f \in [1, m_{fs}]$ **do**
    $Penalty_f = size_f \times (la_{processing,f} + SFMT - LA_f)$;
    if $Penalty_f == 0$ **then**
      if $totalSize + size_f > l_{extra,s}$ **then**
        break;
      $FlowList.append(f, s)$;
      $totalSize = totalSize + size_f$;
// Step 3: Destination NF selection
foreach $d \in NFList_{light}$ **do**
  // Place flows to instance d using Eq. (22),
  // while ensuring $load_d$ does not exceed l_{avg};
  foreach $f \in FlowList$ **do**
    if *flow f should be placed on instance d* **then**
      $migration_{plan}.append(f, s, d)$;

A.4.5 混合场景方案设计

最后，我们讨论当两个或三个 NFV 弹性控制情况同时出现时，OFM 控制器如何决策。我们的策略很简单：当多种情况共存时，我们首先努力使网络运行良

好，然后考虑关闭虚拟机的好处。首先，如果有 NF 过载。我们选择首先处理 NF 过载，因为 NF 过载可能导致高 SLA 违规惩罚。具体来说，我们将流量从负载较重的实例移动到负载较轻的实例，这可以在一定程度上缓解欠载和负载不平衡情况。如果没有轻载实例，我们会创建新的 NF。在很好地处理 NF 过载之后，如果欠载和负载不平衡情况在 NFV 网络中共存，我们需要决定首先处理哪种情况。NF 负载平衡可防止潜在的 NF 过载情况。但是，NF 缩放合并实例以获得收益，并可能增加负载较重的实例的负载，从而导致更高的 NF 过载可能性。因此，我们首先选择平衡 NF 负载。请注意，平衡 NF 负载的过程不会导致 NF 过载。在负载平衡之后，如果任何实例的处理负载低于th_{low}，则 OFM 将执行 NF 收缩并回收没用的虚拟机。总之，三种 NFV 弹性控制情况的反应顺序为：过载→负载不均衡→低载。

A.5 实验验证与分析

A.5.1 实验部署与设置

我们在 Floodlight[34]控制器之上实现了 OFM 控制器。具体来说，我们在一个简单的键值存储数据结构中维护流 SLA，并暴露出其可用于动态附加，修改和删除 SLA 的 REST 接口。NF 状态收集和流信息收集模块在运行时通过 OpenFlow 接口收集 NF 负载和流统计信息，扩展条件触发检测模块使用它来检测 NFV 弹性控制的情况。需要特别注意的是，对于 OFM 的部署，不需要修改 OpenFlow。OFM 只需要交换机能够根据流表规则报告流统计信息和转发流。但是，NF 向控制器公开状态管理接口，以实现安全有效的状态迁移[5]等等。队列开销分析模块计算所需的缓冲成本，扩缩容代价分析模块计算不同情况下的迁移成本。然后，最优迁移方案计算模块将使用第 A.4 节中提供的算法计算优化的流量的集合。为了求解在不同情况下进行 NF 缩放的 ILP，我们使用 lpsolve，一个基于 Java 的混合整数线性规划求解器[35]，来进行求解。

A.5.2 实验结果

我们在基于具有 10 台服务器的测试平台上评估 OFM，每台服务器配备两个 Intel® Xeon® E5-2690v2 CPU。每个 CPU 主频均为 3.00GHz，具有 10 个物理核。同时每台服务器还配有 256G 内存和两个 10G 网卡。服务器运行 Linux 内核版本 4.4.0-31。我们使用一个服务器来运行 OFM Controller，一个用于 Open vSwitch

（OVS）[36]的服务器，以及八个服务器用于 8 个相同类型的 NF 实例。为了避免因虚拟化而影响性能并证明 OFM 在真实物理设备上的可行性，每个软件 NF 都在没有虚拟机或 Docker 封装的裸金属服务器上运行，并且所有服务器都连接到 Pica8 P-3922 的物理交换机。然而，在现实世界的虚拟化环境中，运营商经常使用虚拟机来装在软件 NF，并且多个虚拟机可以共同定位在同一物理服务器中。在这种情况下，虚拟交换机（如 Open vSwitch）可以被用于服务器内的数据包传送[26,27]。我们可以使用虚拟交换机来收集流静态并处理跨多个相同实例的流分布。

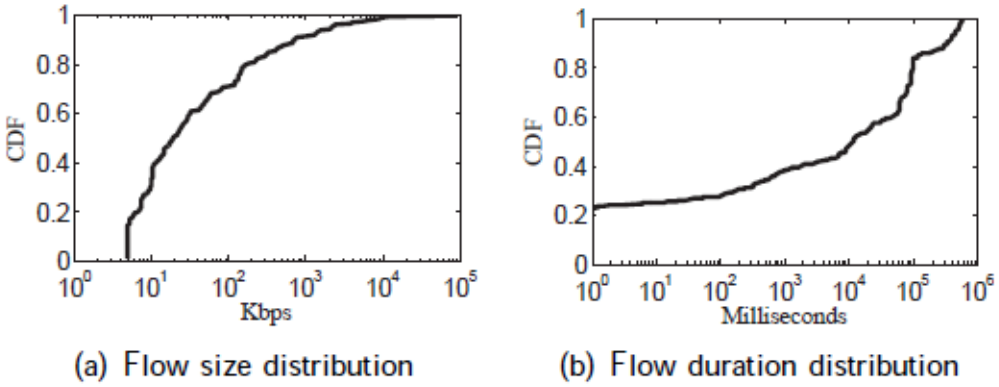


图 A.5 LBNL/ICSI 流量统计（按照源 IP-目的 IP 分开）

关于测试流量，我们使用基于 DPDK 的数据包生成器，该数据包生成器在第五台服务器上运行，并直接连接到承载 OVS 的服务器。生成器发送和接收流量以测量转发延迟。我们使用两种类型的流量模式，包括（1）真实流量：我们使用 LBNL/ICSI 企业流量[37]，这是从现实世界企业网络收集的一组典型流量，其流量大小分布和流持续时间分布如图 A.5 所示；和（2）随机生成的流量，其中我们创建具有随机源和目标地址的流。我们将生成器配置为根据模式类型，流数和流大小创建流量。

在测试 OFM 时，我们有以下目标：

- 证明流迁移时间与要迁移的流数之间的关系。这证明了 OFM 对流量迁移延迟的估计（§A.7.2）。
- 证明在运行时测量期间 OFM 状态收集的及时性和实用性（§A.7.3）。
- 证明 OFM 扩展算法可以找到有效缓解热点的优化迁移计划（§A.7.4）。
- 证明算法中的 OFM 扩缩容可以找到最佳迁移计划，从而带来最大的迁移优势（§A.7.5）。
- 证明 OFM 负载均衡算法有效缓解负载不平衡情况的能力（§A.7.6）。

- 证明 OFM 算法可以在有限的计算时间内有效地解决问题，并且可以在现实世界的网络规模中进行扩展（§ A.7.7）。

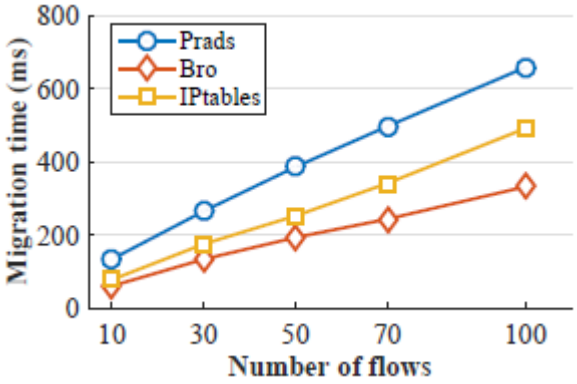


图 A.6 迁移时间和流条数之间的关系

1) 流量迁移时间：在本实验中，我们检查流迁移时间$la_{migration}$和要迁移的流数n之间的关系。我们在两台服务器上启动两个相同类型的 NFs 实例。我们随机生成并向其中一个 NF 实例发送不同数量的流以在其中创建初始流状态。然后我们配置 OFM 控制器来执行此实例上所有流的流和状态迁移到另一个免费实例，并测量迁移时间。我们测试了三种类型的 NF，包括 Prads[38]，Bro[39]和 Iptables[40]。Prads 维护流元数据，终端主机操作系统和服务细节的状态。Bro 维护 TCP，UDP 和 ICMP 的连接信息。IPtables 跟踪所有活动流的 5 元组，TCP 状态，安全标记，等等。对于每种 NF 类型，我们将要迁移的流量从 10 变为 100，并随机改变流速。评估结果如图 A.6 所示，无论流速如何，迁移时间与迁移流量之间呈线性正相关。此外，我们给出了每种 NF 类型的线性回归结果。

- Prads： $la_{migration} = 86.982 + 5.7892 \times n, R^2 = 0.998$
- Bro： $la_{migration} = 39.205 + 2.9545 \times n, R^2 = 0.997$
- IPtables： $la_{migration} = 32.595 + 4.5222 \times n, R^2 = 0.998$

R^2是拟合精度的度量，值为 1 表示线性度很高。上述回归表达式表明迁移时间与迁移流的数量之间存在强烈的线性相关性，可用于估计迁移延迟。特别是，我们可以使用回归表达式来估计 SFMT（指定$n = 1$），这可以用于收缩和负载平衡算法。

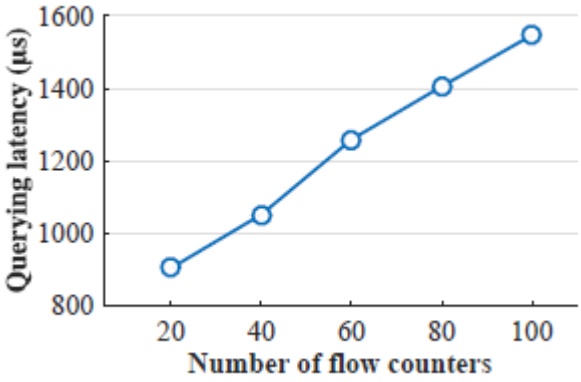


图 A.7 OFM 控制器查询计数器的延时

2) 系统实时性分析：OFM 收集流统计信息并在运行时计算迁移计划。为了确保及时性，必须快速收集统计数据，并且算法应该有效运行。我们使 OFM 控制器能够从一个底层交换机查询不同数量的流量计数器。如图 A.7 所示，OFM 控制器可以在 1.5ms 内获取 100 个计数器。此外，如本节其余部分所示，统计收集和计算的整个控制循环可在 1 秒内完成，这证明了 OFM 的及时性和实用性。

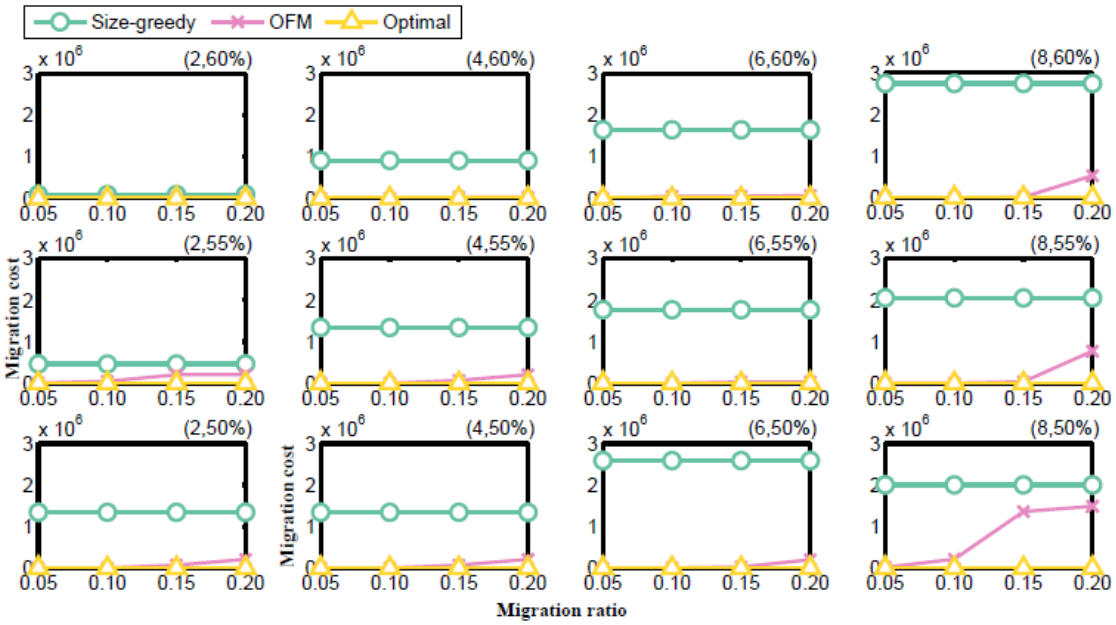


图 A.8 OFM 扩展算法的效果。我们在子图的右上角标注$(\#overloaded\ instances, \mathrm{th}_{safe})$。

3) 动态扩展方案评估：我们使用 Prads NF 评估 NF 缩小算法的优化效果和计算时间。为了模拟 NF 过载情况，我们假设有 10 个 NF 实例，并将重载实例的数量设置为 2，4，6，8。随着重载实例的数量增加，OFM 扩展算法在新实例部署阶段更有可能被触发，这会带来更高的惩罚。我们将th_{safe}分别设置为 60%，55%和

50%，并假设在每种情况下，流量分别为 5%，10%，15%和 20%需要为每个重载实例迁移（大小）。th_{safe}的减少表示当前实例可以容纳更少的流，并且更有可能部署新实例。为了量化迁移成本，由于缺乏 NFV 网络的实际 SLA 设置，我们通过遵循$[0.5 \times (SFMT + la_{processing}), 1.5 \times (SFMT + la_{processing})]$均匀随机分布来设置流的 SLA。这可以确保在迁移期间违反某些流 SLA，而有些则不会。

我们将 OFM 扩展算法与最优算法和按流尺寸贪心算法进行比较。最优解决方案可生成流量集，以最小的成本覆盖足够的流量以进行迁移。对于尺寸贪心的解决方案，它选择具有最大尺寸的流量，直到选择足够的流量来缓解热点，从而使流量选择过程尽可能快地完成。对于把流放到实例上这一步骤，它选择大流量放在当前实例上的小流量上。这里的直觉是最大化当前实例上的流的大小。最后，对于新实例部署过程，尺寸贪心策略始终将大型流包装到新实例中，以便最小化新实例的数量。如图 A.8 所示，与尺寸贪婪算法相比，OFM 扩展算法可以在很大程度上降低迁移成本，同时与最优解相比成本略高一点点。这证明了 OFM 算法的有效性。

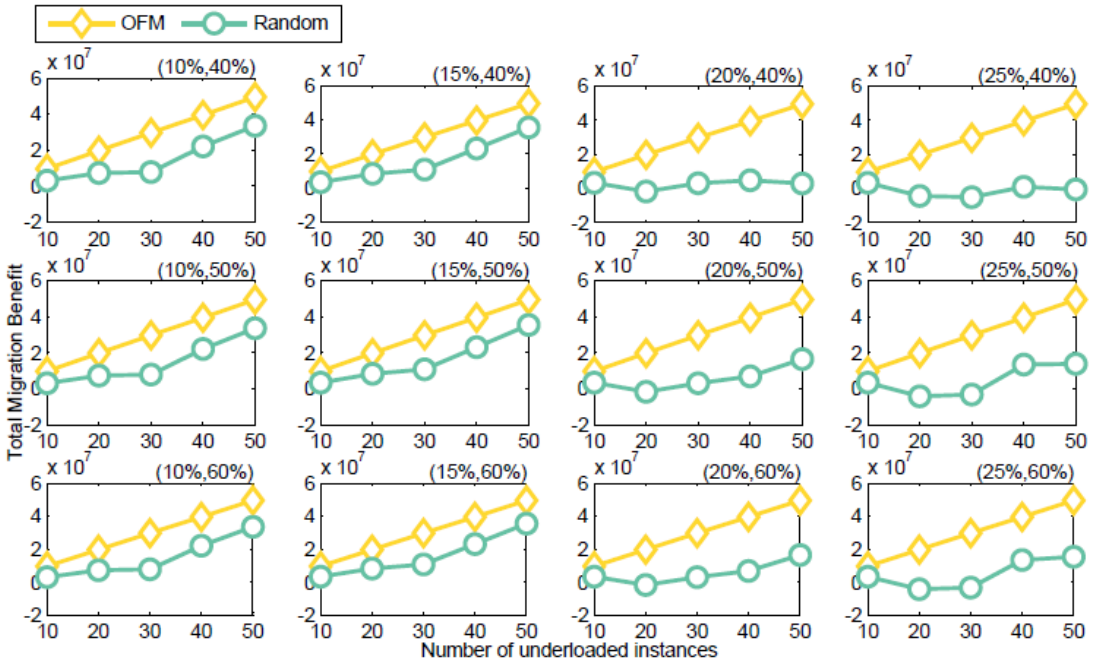


图 A.9 OFM 收缩算法的效果。我们在子图的右上角标注(th_{bottom}, th_{safe})。

4) 动态收缩方案评估：OFM 利用 ILP 计算最佳解决方案，该解决方案可以最大限度地降低 NF 缩放的迁移成本。为了评估优化效果，我们将th_{bottom}设置为 10%，15%，20%，25%和th_{safe}为 40%，50%，60%。在运行时期间，操作员可以动态地配置高于阈值。我们将流从 LBNL/ICSI 企业流量分散到 NF 实例，以确保

一定数量的 NF 实例欠载。我们按照与 NF 扩展实验相同的均匀随机分布配置流的 SLA。

这种方法的表现几乎完全取决于 ILP 的制定和解决方案。ILP 性能主要受相同类型的欠载 NF 实例数量的影响。我们将欠载实例的数量设置为 100 个实例总数中的 10 个，20 个，30 个，40 个和 50 个。我们使用 Prads NF 进行评估。我们将 OFM 中算法中的 NF 缩放与随机解决方案进行比较，该随机解决方案随机选择 NF 对进行合并，同时确保合并后的总 NF 负载不超过th_{safe}。如图 A.9 所示，OFM 解决方案可以通过增加欠载实例数来实现迁移优势的线性增加，并且总是显着优于随机解决方案。

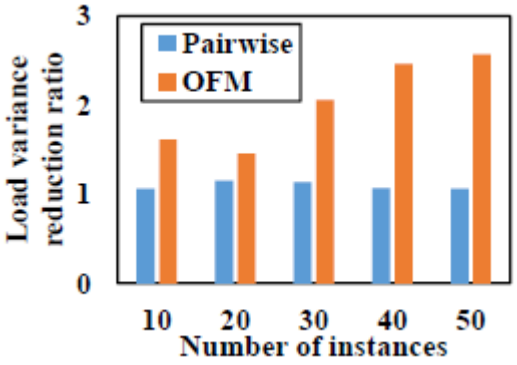


图 A.10 OFM NF 负载均衡算法效果

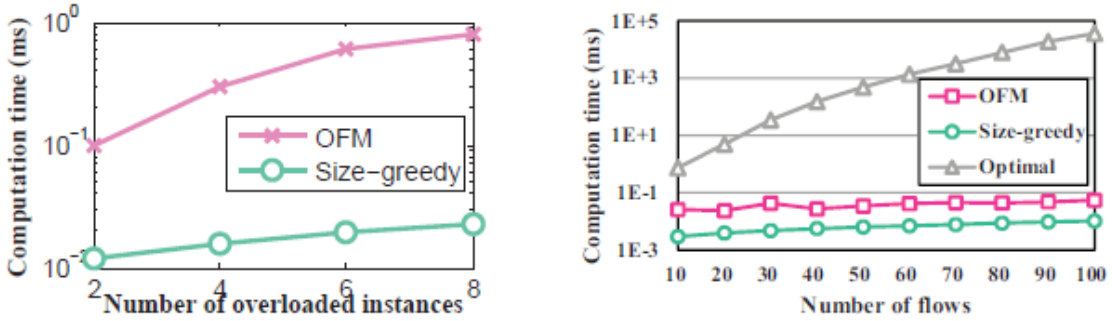
5) 负载均衡方案评估：OFM 中的 NF 负载平衡目标是减少属于同一 NF 类型的 NF 实例的负载方差。因此，我们将 NF 实例的数量从 10 改为 50，计算 NF 实例的之前（var_{before}）和之后（var_{after}）的负载，并计算方差缩减倍率$ratio = var_{before}/var_{after}$。我们从 NF 实例上的 LBNL/ICSI 企业跟踪中随机排列流，以确保不会发生过载或欠载情况。我们将 OFM 中的 NF 负载平衡算法与逐对解决方案进行比较，该解决方案通过对 NF 实例的负载进行排序并迭代地选择具有最低和最高负载的实例作为对来贪婪地配对重载和欠载的 NF 实例。然后，它在每对中的两个实例之间重新分配流以进行负载平衡。如图 A.10 所示，OFM 负载均衡算法可以将 NF 实例的负载方差减少 1.5 到 2.5 倍，即 20%到 60%比逐对解决方案更好。

6) 系统可扩展性分析：我们评估 OFM 相对于真实世界网络参数的可扩展性。根据 OFM 算法的设计，流条数，包速率和实例个数可能会影响计算时间。由于包速率可以根据实际网络流量的流量大小分布[2]轻松转换为流条数，下面我们将重点放在 OFM 关于流条数和实例个数的可扩展性。对于我们实验中的测试流量，

我们使用来自 LBNL/ICSI 企业流量，其统计数据如图 A.5 所示。我们在表 A.2 中总结了 OFM 的可扩展性评估。

表 A.2 OFM 算法与不同参数的可扩展性

Parameter	Scale out	Scale in	Load balancing
流条数	如图 6.12所示.	无关.	如图 6.11所示.
包速率	基于真实世界流量分布，包速率可被转化为流条数[2].		
实例个数	如图 6.12所示	如图 6.13所示	如图 6.11所示



(a) With different number of instances. (b) With different number of flows.

图 A.11 OFM 扩展算法

对于 OFM 扩展算法，我们首先使用与 A.5.4 节相同的评估设置，在不同数量的实例下测量计算时间。如图 A.11 所示，OFM 算法消耗的计算时间少于 1ms，这只占整个迁移时间的一小部分。尽管大小贪婪的算法可以在 0.1ms 内完成更快，但其优化效果远远落后于 OFM，如 A.5.4 节中所述。其次，我们改变一个实例上的流数量，并将实例数量设置为一个。如图 A.11 所示，OFM 的计算时间在不同数量的流量下低于 0.1ms，并随着流量的增加而缓慢增加。此外，OFM 计算时间明显短于最优解。以上结果证明了 OFM 扩展算法的可扩展性和效率。

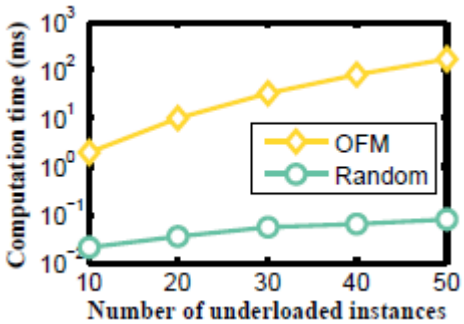
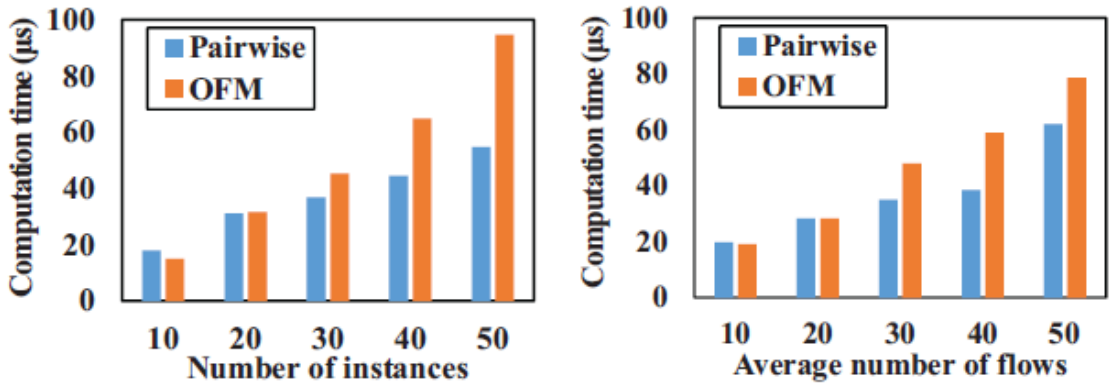


图 A.12 OFM 收缩算法

对于 OFM 收缩算法，由于算法设计中与流的数量无关，因此我们改变欠载实例的数量以评估其可伸缩性。根据图 A.12，当处理 10 到 50 个欠载实例时，算法中 OFM NF 缩放的计算时间低于 200ms，这在现实世界中是可接受。



(a) With different number of instances. (b) With different number of flows.

图 A.13 OFM NF 负载均衡算法可扩展性

对于 OFM 负载均衡算法，我们首先将每个实例上的平均流量数设置为 10，改变实例数量，并测量计算时间。然后，我们将实例数设置为 10，改变每个实例上的平均流数，并测量计算时间。如图 A.13 所示，OFM 算法的计算时间远低于 100μs所有参数配置，演示了算法的可扩展性，以快速平衡现实世界 NFV 网络中的 NF 负载。

6. 相关工作

一些研究工作已经强调了状态迁移支持 NFV 弹性控制的必要性[5,12-16,41]。Split/Merge[14]和 OpenNF[5]依赖于集中控制平面来缓冲迁移期间的状态，而加强版的 OpenNF[12]和一些其他工作[13,15,16,41]完全在数据平面中执行状态和数据包传输，以提高可伸缩性和性能。以上工作主要集中在 NFV 中的安全有效状态迁移。相比之下，OFM 解决了优化流选择对 NFV 弹性控制的挑战，以最大限度地减少惩罚，并且是对上述工作的补充。

Kablan 等人[42]建议从 NF 中提取状态并将状态存储在数据存储层中，从而消除了迁移 NFV 弹性控制流的必要性。但是，这样的设计可能会使 NF 处理延迟增加最多 500μs，这对延迟敏感的应用程序来说可能无法忍受[16,17]。相比之下，OFM 仔细考虑流量的 SLA 要求，并选择适当的流量进行迁移，以实现优化的 NFV 弹性控制。

在 E2[18]中提出的用于 NFV 弹性控制的简单解决方案采用了避免迁移的策略。现有流仍然由先前分配的 NF 实例处理，而新流被差分处理。这样，NFV 弹性控制不会发生流动迁移。对于 NF 扩展，我们只是实例化一个新的 NF 实例并将新流重定向到它。对于 NF 收缩，我们在几个选定的 NF 上合并新流，并在提供所有剩余流后终止其他服务器。对于 NF 负载平衡，我们利用一致的哈希来平衡新流。虽然迁移避免策略不会引入迁移惩罚，但仍可能导致惩罚。对于 NF 扩展，现有 NF 实例上的流量可能会增大，从而增加 NF 负载，降低 NF 性能，并导致 SLA 被违反。对于 NF 收缩，数据中心中的许多流的流完成时间很长，可能持续数分钟到数小时[3]。迁移避免策略可防止及时销毁负载不足的实例，因此无法带来与 OFM 一样高的收益。对于 NF 负载平衡，当现有 NF 实例上的流量大小增加时，NF 实例可能会变得过载并触发 NF 扩展，这也会在没有仔细选择流量的情况下引入 SLA 违规惩罚。

流迁移，也称为流切换，也是 5G 网络中的一个重要问题。最近的两个典型研究表明了从 5G 宏小区网络到小小区的流量接纳控制的必要性。[43,44]已经说明了这种研究的必要性。以上工作旨在最小化流程切换期间对其他用户的体验的影响。同样，OFM 旨在最小化 NF 弹性控制和流量迁移期间的 SLA 违规惩罚。但是，OFM 与上述作品有很大不同。对于 5G 网络，宏小区网络本身可以处理所有分组，而采用小小区来降低功率和成本。因此，将流量迁移到小型小区是可选的，并且需要准入控制机制来确保用户体验。但是，在 NFV 网络中，一个 NF 实例通常不足以处理所有流量，从而使弹性控制不可避免。OFM 可以优化迁移惩罚。

最后，与本文的较早版本相比[1]，我们在这一文章中做了质的改变。第一，我们为三种 NFV 弹性控制场景提供了统一的 ILP 建模，来产生最优流迁移计划。然而，在不违背 NFV 弹性控制场景的情况下，最优计划在给定时间内难以求解。因此，我们需要为三种场景分别设计算法。第二，我们彻底地修改了三种流迁移场景下的最优算法。我们首先为每个场景提出了其最优方案，其次用来有效、高效求解的加速方案。此外，我们讨论了当两个或三个 NFV 弹性控制情况同时发生时，OFM 如何反应。最后，我们更新了评估，以证明 OFM 可以通过优化的流量迁移实现 NFV 弹性控制。

7. 结论

我们提出了 OFM 控制器的设计，以实现 NFV 弹性控制的优化流量迁移。我们分析了不同的 NFV 弹性控制情况，包括 NF 缩小，缩放和负载平衡，并确定了他们的控制目标和挑战。在对缓冲区队列和迁移成本进行建模后，我们针对所有三种情况引入了统一的最优公式，这在有限的时间内无法解决。因此，为了实现每种情况的独特控制目标，我们针对每种情况设计了独特的最优公式和快速启发式算法。最后，我们还介绍了 OFM 选择在 NFV 中共存两三种情况时按顺序处理 NF 过载，负载不平衡和 NF 欠载。我们在 NFV 和 SDN 环境之上实现了 OFM 控制器。广泛的评估结果表明，OFM 可以在合理的计算时间内实现接近最优的流量迁移。

参考文献

[1] C. Sun, J. Bi, Z. Meng, X. Zhang, and H. Hu, “OFM: Optimized flow migration for NFV elasticity control,” in Proc. IEEE/ACM 21st Int. Symp. Qual. Service (IWQoS), Jun. 2018, pp. 1–10.

[2] R. Guerzoni, “Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action, introductory white paper,” in Proc. SDN OpenFlow World Congr., 2012.

[3] T. Benson, A. Akella, and D. A. Maltz, “Network traffic characteristics of data centers in the wild,” in Proc. 10th ACM SIGCOMM Conf. Internet Meas., 2010, pp. 267–280.

[4] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, “Inside the social network’s (datacenter) network,” ACM Comput. Commun. Rev., vol. 45, no. 4, pp. 123–137, 2015.

[5] A. Gember-Jacobson et al., “OpenNF: Enabling innovation in network function control,” in Proc. ACM Conf. SIGCOMM, 2014, pp. 163–174.

[6] P. Demestichas et al., “5G on the horizon: Key challenges for the radioaccess network,” IEEE Veh. Technol. Mag., vol. 8, no. 3, pp. 47–53, Sep. 2013.

[7] B. Blanco et al., “Technology pillars in the architecture of future 5G mobile networks: NFV, MEC and SDN,” Comput. Standards Interfaces, vol. 54, pp. 216–228, Nov. 2017.

[8] W. Haeffner, J. Napper, M. Stiemerling, D. R. Lopez, and J. Uttaro, “Service function chaining use cases in mobile networks,” in Proc. Internet Eng. Task Force, 2016, pp. 1–26.

[9] S. Dutta, T. Taleb, and A. Ksentini, “QoE-aware elasticity support in cloud-native 5G systems,” in Proc. IEEE Int. Conf. Commun. (ICC), May 2016, pp. 1–6.

[10] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, “Openstate: Programming platform-independent stateful openflow applications inside the switch,” ACM SIGCOMM Comput. Commun. Rev., vol. 44, no. 2, pp. 44–51, 2014.

[11] M. Moshref, A. Bhargava, A. Gupta, M. Yu, and R. Govindan, “Flowlevel state transition as a new switch primitive for SDN,” in Proc. ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN), 2014, pp. 61–66.

[12] A. Gember-Jacobson and A. Akella, “Improving the safety, scalability, and efficiency of network function state transfers,” in Proc. ACM SIGCOMM Workshop Hot Topics Middleboxes Netw. Function Virtualization, 2015, pp. 43–48.

[13] B. Kothandaraman, M. Du, and P. Sköldström, “Centrally controlled distributed VNF state management,” in Proc. ACM SIGCOMM Workshop Hot Topics Middleboxes Netw. Function Virtualization, 2015, pp. 37–42.

[14] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, “Split/Merge: System support for elastic execution in virtual middleboxes,” in Proc. 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI), 2013, pp. 227–240.

[15] Y. Wang, G. Xie, Z. Li, P. He, and K. Salamatian, “Transparent flow migration for NFV,” in Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP), Nov. 2016, pp. 1–10.

[16] S. Woo, J. Sherry, S. Han, S. Moon, S. Ratnasamy, and S. Shenker, “Elastic scaling of stateful network functions,” in Proc. NSDI, 2018, pp. 299–312.

[17] R. Gandhi et al., “Duet: Cloud scale load balancing with hardware and software,” in Proc. ACM Conf. SIGCOMM, 2014, pp. 27–38.

[18] S. Palkar et al., “E2: A framework for NFV applications,” in Proc. 25th Symp. Operating Syst. Princ., 2015, pp. 121–136.

[19] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, “Making middleboxes someone else’s problem: Network processing as a cloud service,” ACM SIGCOMM Comput. Commun. Rev., vol. 42, no. 4, pp. 13–24, 2012.

[20] M. Alhamad, T. Dillon, and E. Chang, “Conceptual SLA framework for cloud computing,” in Proc. 4th IEEE Int. Conf. Digit. Ecosyst. Technol. (DEST), Apr. 2010, pp. 606–610.

[21] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic flow scheduling for data center networks,” in Proc. NSDI, vol. 10, 2010, p. 19.

[22] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “DevoFlow: Scaling flow management for highperformance networks,” Comput. Commun. Rev., vol. 41, no. 4, pp. 254–265, Aug. 2011.

[23] T. Benson, A. Anand, A. Akella, and M. Zhang, “MicroTE: Fine grained traffic engineering for data centers,” in Proc. 7th Conf. Emerg. Netw. Exp. Technol., 2011, p. 8.

[24] Openflow Switch Specification 1.4.0, Open Netw. Found., 2013.

[25] S. Miteff and S. Hazelhurst, “NFShunt: A Linux firewall with openflow-enabled hardware bypass,” in Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN), Nov. 2015, pp. 100–106.

[26] J. Martins et al., “ClickOS and the art of network function virtualization,” in Proc. 11th USENIX Symp. Netw. Syst. Design Implement. (NSDI), Seattle, WA, USA, 2014, pp. 459–473.

[27] J. Hwang, K. K. Ramakrishnan, and T. Wood, “NetVM: High performance and flexible networking using virtualization on commodity platforms,” IEEE Trans. Netw. Service Manage., vol. 12, no. 1, pp. 34–47, Mar. 2015.

[28] N. Foster et al., “Frenetic: A network programming language,” ACM SIGPLAN Notices, vol. 46, no. 9, pp. 279–291, 2011.

[29] L. Wu, S. K. Garg, and R. Buyya, “SLA-based resource allocation for software as a service provider (SAAS) in cloud computing environments,” in Proc. 11th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGrid), May 2011, pp. 195–204.

[30] J. P. Ignizio, Goal Programming and Extensions. Lanham, MD, USA: Lexington Books, 1976.

[31] A. Schrijver, Theory of Linear and Integer Programming. Hoboken, NJ, USA: Wiley, 1998.

[32] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, “NFP: Enabling network function parallelism in NFV,” in Proc. Conf. ACM Special Interest Group Data Commun., 2017, pp. 43–56.

[33] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, “Silkroad: Making stateful layer-4 load balancing fast and cheap using switching ASICs,” in Proc. Conf. ACM Special Interest Group Data Commun., 2017, pp. 15–28.

[34] Project Floodlight. Accessed: Aug. 28, 2018. [Online]. Available: http://www.projectfloodlight.org/floodlight

[35] M. Berkelaar, J. Dirks, K. Eikland, P. Notebaert, and J. Ebert. (2007). Lpsolve: A Mixed Integer Linear Programming (MILP) Solver. [Online]. Available: http://sourceforge.net/projects/lpsolve

[36] Open vSwitch. Accessed: Aug. 28, 2018. [Online]. Available: http://openvswitch.org

[37] LBNL/ICSI Enterprise Tracing Project. Accessed: Aug. 28, 2018. [Online]. Available: http://www.icir.org/enterprise-tracing

[38] Passive Real-Time Asset Detection System. Accessed: Aug. 28, 2018. [Online]. Available: http://gamelinux.github.io/prads/

[39] V. Paxson, “Bro: A system for detecting network intruders in real-time,” Comput. Netw., vol. 31, nos. 23–24, pp. 2435–2463, 1999.

[40] Netfilter/Iptables Project. Accessed: Aug. 28, 2018. [Online]. Available: http://www.netfilter.org

[41] Y. Lin, U. C. Kozat, J. Kaippallimalil, M. Moradi, A. C. K. Soong, and Z. M. Mao, “Pausing and resuming network flows using programmable buffers,” in Proc. SOSR, 2018, p. 7.

[42] M. Kablan, A. Alsudais, E. Keller, and F. Le, “Stateless network functions: Breaking the tight coupling of state and processing,” in Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI), Boston, MA, USA, 2017, pp. 97–112.

[43] T. Taleb and A. Ksentini, “QoS/QoE predictions-based admission control for femto communications,” in Proc. IEEE Int. Conf. Commun. (ICC), Jun. 2012, pp. 5146–5150.

[44] A. Ksentini, T. Taleb, and K. B. Letaif, “QoE-based flow admission control in small cell networks,” IEEE Trans. Wireless Commun., vol. 15, no. 4, pp. 2474–2483, Apr. 2016.

附录 B 恒定带宽链路上的自适应比特率调整算法

在本章中，我们为 4.5.1 节中的将链路带宽固定为 3000kbps 和 1300kbps 的实验提供更多细节分析。

B.1 3000kbps 链路用户体验分析

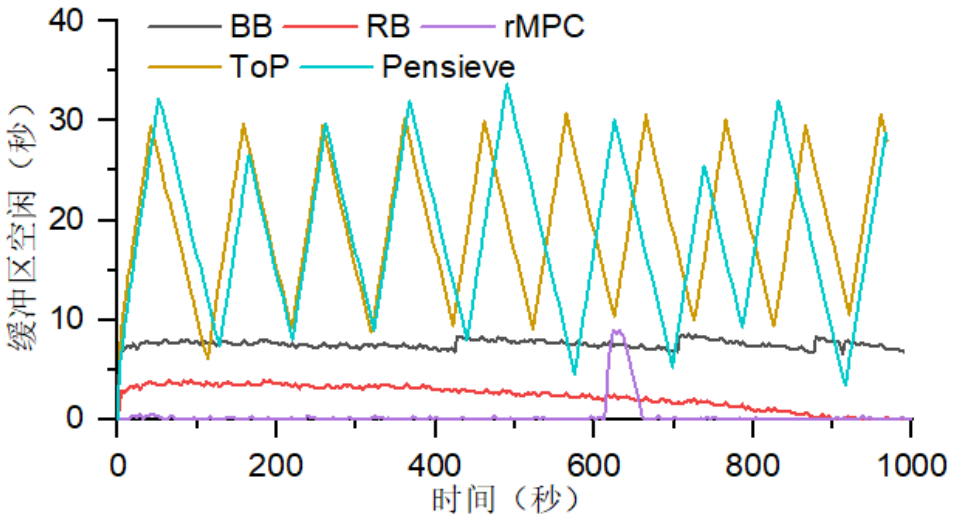


图 B.1 3000kbps 链路上的缓冲区空闲情况

除了 4.5.1 节中的实验外，我们还测量了 3000kbps 链路上的运行时缓冲区空闲情况。如图 B.1 所示，Pensieve 的缓冲区空闲情况不断波动：当 720p 被选择时，缓冲区空闲增长；当 1440p 被选择时，缓冲区空闲减少。这也被 ToP 保真地模仿到了。这导致了严重的平滑性上的惩罚。同时，缓冲区空闲情况也能解释中的 rMPC 性能较差的原因：rMPC 在一开始就收敛了，由于每个视频块的大小并不完全相同，因此其并没有足够的缓冲区来应对视频块大小的波动。因此 rMPC 有较高的卡顿惩罚。在 1000 秒的实验中，BB 和 RB 的缓冲区也轻微下降，这是因为有效带宽并不恰好是 2850kbps（所采用视频的平均比特率）。

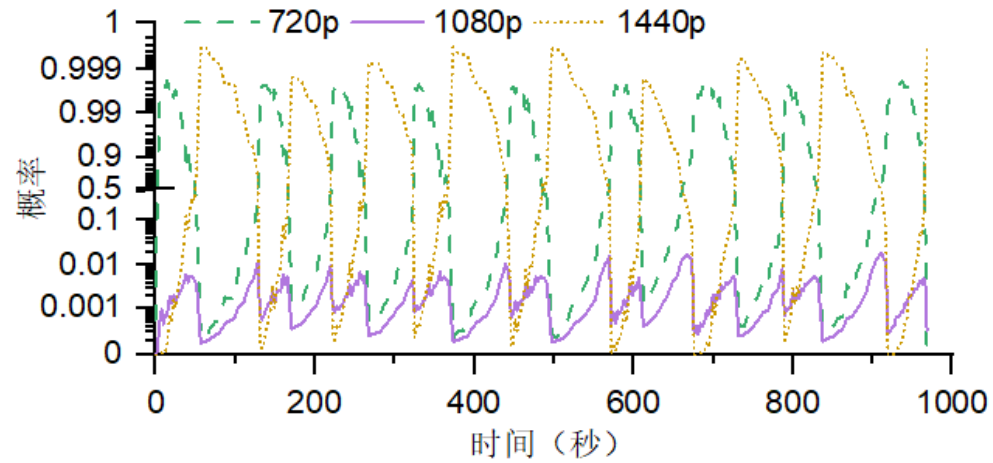


图 B.2　选择 720p、1080p、1440p 的概率。选择其他三种比特率的概率由于小于10^{-4}因而并未画出。

由于 Pensieve 的神经网络的原始输出是选择每个比特率的归一化的概率，我们进一步分析 Pensieve 在选择不同比特率上的概率，结果如图 B.2 所示。接近于 1 的比特率表明在决策上具有更高的信心。我们发现，Pensieve 在它所做的决策上并没有足够的信心，这可能意味着 Pensieve 在训练时可能并未经历过类似情况，因而他并不知道如何在这种情况下决策。

B.2 1300kbps 链路用户体验分析

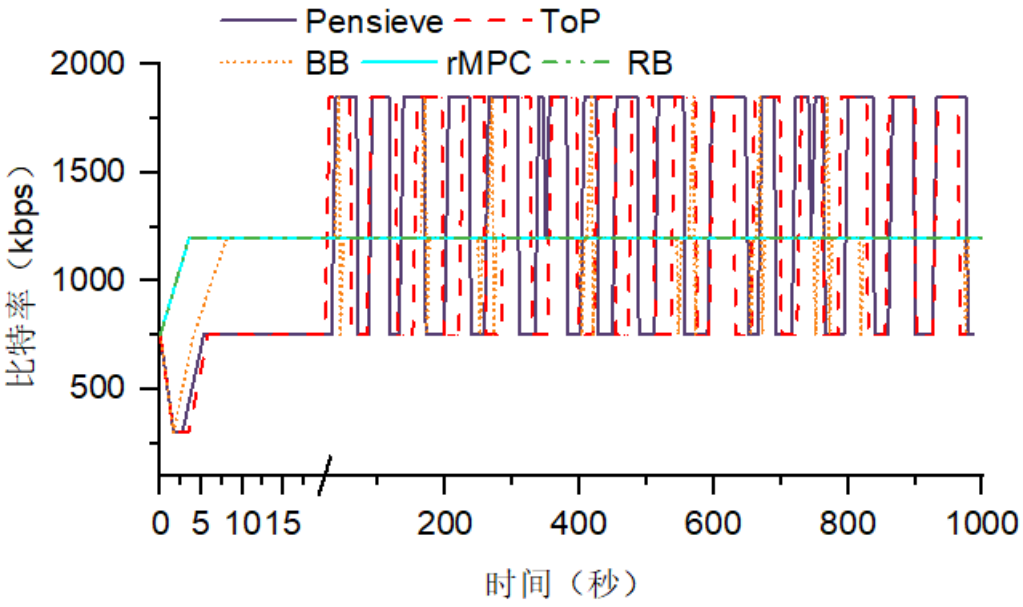


图 B.3　1300kbps 恒定链路上的比特率变化情况

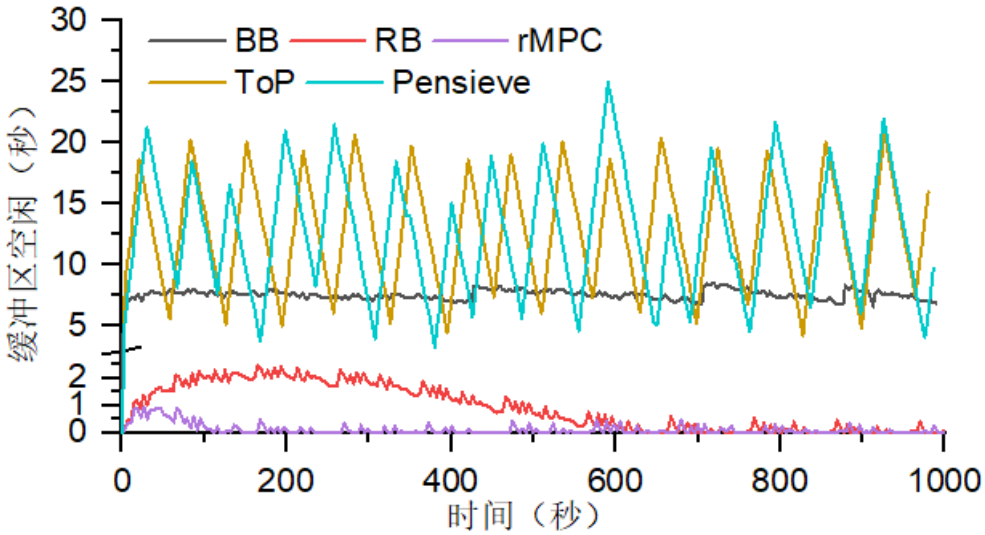


图 B.4　1300kbps 恒定链路上的缓冲区变化情况

表 B.1　1300kbps 上的用户体验结果

BB	RB	rMPC	ToP	Pensieve
1.050	0.904	0.803	0.986	0.983

我们同样为在图 4.14 中，1300kbps 链路上的实验在此描述一些细节。结果如图 B.3、图 B.4 和表 B.1 所示。1300kbps 链路下的实验结果与 3000kbps 链路下的结果相似。需要特殊注意的是，RB 由于其收敛较快，因此性能较差。

在学期间参加课题的研究成果

[1] **Zili Meng**, Jun Bi, Chen Sun, Shuhe Wang, Minhu Wang, Hongxin Hu. PAM: When Overloaded, Push Your Neighbor Aside!, in proceedings of *ACM SIGCOMM Posters and Demos*, Budapest, Hungary, 2018.

[2] **Zili Meng**, Jun Bi, Haiping Wang, Chen Sun, Hongxin Hu. CoCo: Compact and Optimized Consolidation of Modularized Service Function Chains in NFV, in proceedings of *IEEE International Conference on Communications (ICC)*, Kansas City, USA, 2018.

[3] **Zili Meng**, Jun Bi, Chen Sun, Anmin Xu, Hongxin Hu. PRAM: Priority-aware Flow Migration Scheme in NFV Networks, in proceedings of *ACM Symposium on Software-Defined-Networking Research (SOSR, poster)*, Santa Clara, USA, 2017.

[4] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, **Zili Meng**, Mohammad Alizadeh. Learning Scheduling Algorithms for Data Processing Clusters, accepted by *ACM SIGCOMM*, Beijing, China, 2019.

[5] Chen Sun, Jun Bi, **Zili Meng**, Tong Yang, Xiao Zhang, Hongxin Hu, Enabling NFV Elasticity Control with Optimized Flow Migration, in *IEEE Journal on Selected Areas in Communications (JSAC)*, Vol.36, No. 10, pp2288-2303, 2018.

[6] Chen Sun, Jun Bi, **Zili Meng**, Xiao Zhang, Hongxin Hu. OFM: Optimized Flow Migration for NFV Elasticity Control, in proceedings of *IEEE/ACM International Symposium on Quality of Service (IWQoS)*, Banff, Canada, 2018.

[7] Zhilong Zheng, Jun Bi, Chen Sun, Heng Yu, Hongxin Hu, **Zili Meng**, Shuhe Wang, Kai Gao, Jianping Wu. GEN: A GPU-Accelerated Elastic Framework for NFV, in proceedings of *Asia-Pacific Workshop on Networking (APNet)*, Beijing, China, 2018.

综合论文训练记录表

学生姓名	孟子立	学号	2015011189	班级	无 58
论文题目	智能网络系统的实用性与可解释性研究				
主要内容以及进度安排	本论文研究智能网络系统的实用性和可解释性，并提出了 TranSys 方法，通过将基于深度强化学习的智能网络系统中的深度神经网络高保真度地转换为决策树，来缩短决策延迟、降低资源消耗，并提升系统可解释性。由于可解释性和实用性的研究刚刚展开，机器学习在网络系统中的应用都还不多，本文通过在多个系统上测试方法的性能，为该领域研究提供方法、数据支持和一定的方向指导。 进度安排： 2018/11-2018/12：对智能网络系统的应用情况进行文献调研，具体总结其存在的实用性问题和可解释性问题。 2018/12-2019/1：初步提出通过线性拟合等方式来对深度神经网络进行转换，并进行初步实验以比选方案。 2019/2-2019/3：搭建实验网络拓扑、配置实验环境，搭建模型转换系统，实现设计思路。 2019/4-2019/5：进行性能评估与对比实验，并不断根据结果及实验中发现的问题迭代设计 2019/5-2019/6：总结模型与实验数据，完成毕业设计论文。 **指导教师签字：** 王东升 **考核组组长签字：** 苏生 2019 年 1 月 11 日				
中期考核意见	孟子立同学的毕设中期检查符合要求，按进度规划后期进度可达毕设基本要求。 **考核组组长签字：** 苏生 2019 年 4 月 19 日				

指导教师评语	论文前期调研充分，研究工作扎实，论证严密，文字流畅，达到学士学位论文水平。 指导教师签字： 2019 年 5 月 22 日
评阅教师评语	论文研究基于深度强化学习的智能网络系统。通过把深度神经网络转换为决策树，缩短决策延迟，提高深度强化学习在智能网络系统中的实用性和可解释性。 论文工作扎实，具有较强的创新性，论文结构完整，叙述清楚，达到学士学位论文水平。 评阅教师签字： 2019 年 6 月 10 日
答辩小组评语	论文研究智能网络的可解释性，选题具有实际价值。论文研究深度神经网络转换决策树的方法，取得较好的进展。答辩回答问题正确，讲述清楚。 答辩小组组长签字： 2019 年 6 月 13 日
	总成绩：A- 教学负责人签字： 2019 年 6 月 21 日