arXiv:2302.04519v2 [cs.NI] 4 Oct 2023

RayNet：用于开发强化学习驱动的网络协议的仿真平台

LUCA GIACOMONI、Basil Benny 和 George Parisis，英国萨塞克斯大学

强化学习（RL）在网络协议开发领域日益受到关注。然而，基于强化学习的协议仍处于起步阶段，要构建可部署的解决方案，还需要开展大量研究。开发基于强化学习的协议是一个复杂且具挑战性的过程，涉及多项模型设计决策，并且需要在真实和仿真网络拓扑中进行大量训练与评估。网络模拟器可为基于强化学习的协议提供高效的训练环境，因为它们具有确定性，且可并行运行。本文介绍*RayNet*，这是一个可扩展、可适配的仿真平台，用于开发基于强化学习的网络协议。RayNet 将完全可编程的网络模拟器 OMNeT++ 与面向分布式强化学习的可扩展训练平台 Ray/RLlib 集成起来。RayNet 有助于系统地开发基于强化学习的网络协议，使研究人员能够专注于当前问题，而不必纠结于研究中学习部分的实现细节。我们开发了一种简单的基于强化学习的拥塞控制方法作为概念验证，展示了 RayNet 可成为计算机网络强化学习研究的实用平台，支持可扩展的训练与评估。我们将 RayNet 与*ns3-gym*进行了比较；该平台与 RayNet 的目标类似。结果表明，在智能体从强化学习环境中收集经验的速度方面，RayNet 表现更佳。

CCS概念： • **计算方法** → **仿真工具**； 序贯决策； • **网络** → 网络协议设计。

补充关键词与短语：计算机网络协议、网络仿真、强化学习、拥塞控制

ACM 引用格式：

Luca Giacomon i、Basil Benny 和 George Parisis。2023年。RayNet：用于开发强化学习驱动的仿真平台网络协议 第 1 卷第 1 期（2023年10月），共 24 页。https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 引言

强化学习（RL）在开发拥塞控制算法 [1, 56],路由 [26], 视频码率控制 [17, 27],网络接入 [33, 52],网络安全 [36, 49] 和主动缓存 [15, 62]方面势头强劲。基于强化学习的算法有望适应各种网络部署、拓扑结构、流量工作负载和链路技术，而无需针对上述多样性设计并调优专用算法。这进而为开发面向未来的算法创造了机会：这些算法可以离线训练，并随着网络和用户工作负载的演变持续优化自身行为。基于强化学习的协议仍处于起步阶段，要产出可部署的解决方案，还需要大量研究 [11]。开发基于强化学习的协议是一个复杂过程，需要：（1）选择一种特定的强化学习算法；（2）针对当前问题设计合适且有效的强化学习模型；（3）在真实的网络环境中训练智能体；（4）将其部署到实际环境中。前两项涉及一系列设计决策，包括强化学习智能体的动作和状态空间、奖励函数以及强化学习

作者地址：Luca Giacomoni，l.giacomoni@sussex.ac.uk；Basil Benny，bb314@sussex.ac.uk；George Parisis，g.parisis@sussex.ac.uk，英国布莱顿萨塞克斯大学。

允许免费制作本作品全部或部分内容的数字或纸质副本，供个人或课堂使用，前提是这些副本不得以营利或商业利益为目的制作或分发，且副本须附有此声明及首页上的完整引文。对于本作品中归 ACM 以外者所有的组成部分，必须尊重其版权。注明出处后可以摘录。若要以其他方式复制、再版、发布到服务器或重新分发至列表，须事先获得具体许可和/或支付费用。请向 permissions@acm.org 申请许可。© 2023年 美国计算机协会。提交给 ACM 的稿件

算法本身。训练智能体绝非易事；需要探索大量超参数（例如折扣率 *gamma*、回放缓冲区大小等），以确保在给定所选强化学习算法和模型、训练设置（例如所用网络参数、拓扑结构和工作负载）以及预期部署参数的情况下，得到尽可能优的策略。同时，训练智能体需要收集海量的智能体经验，这可以通过真实网络（例如 [1]）、网络仿真器（例如 [39]）或网络仿真（例如 [48]）来完成。

我们认为，网络模拟器，尤其是离散事件分组级模拟器，为训练基于强化学习的网络协议及其相应算法提供了非常有效的平台。首先，模拟器提供了完全可控且可配置的实验环境。网络模拟器采用领域专用语言，使定义待仿真的网络（包括底层拓扑结构、链路特性和流量工作负载）变得轻松。此外，各项仿真可以彼此独立并行运行。若要在真实网络中实现这种可配置性，成本和时间开销都将极其高昂；而网络仿真（例如 [35], 中采用的仿真）仅支持有限的可配置性。其次，训练网络协议的智能体需要在多个端点之间交换流量。在真实或仿真的网络中，这些流量必须实际通过网络传输；根据训练设置和参数，这可能会显著增加训练过程的开销，具体表现为收集智能体经验所需的时间变长（例如，在网络容量极小，或智能体尚远未达到最优状态的情况下）。更糟的是，由于智能体需要以远快于训练器计算速度的频率采取动作，为高速网络训练拥塞控制策略可能非常困难（甚至根本无法实现）。[1]相反，低效策略产生的网络流量，或经过低带宽链路的流量，都可以在仿真网络中快速传输；而部署环境的仿真性质则消除了高性能网络部署带来的问题。第三，网络模拟器在设计上支持结果的可复现性，这对基于强化学习的方法至关重要。令人惊讶的是，许多关于基于强化学习的网络技术的论文在发表时，既未提供任何可复现性保障，也未对此进行讨论[2]；而其他研究社区早已将可复现性纳入同行评审流程。真实网络或仿真网络均无法实现可复现性。最后，我们认为，开发训练平台将普遍有益于研究社区；我们设想这样一个平台： [7] 研究人员可以在其中提出问题及相应的强化学习模型、训练智能体，并分享习得策略以及所有选定的参数和超参数，而无需处理将学习功能嵌入仿真模型时涉及的实现细节（以及不可避免的程序错误）。

近年来，我们看到大量关于基于强化学习的网络技术的研究，例如缓存、路由和拥塞控制领域，这些技术的训练与评估均通过仿真完成。虽然人们原本预期会出现能够支持开发上述技术的仿真平台，但大多数文献采用的都是定制方法：要么从头编写（例如 [33, 38]），要么使用现有框架（例如 [55, 57]），但仅适用于特定问题，而且在大多数情况下不向公众开放。缺乏此类公开且适用范围广泛的仿真平台，不利于研究社区的发展。定制方法可能效率低下且存在缺陷；若有一个广泛采用的平台，提供标准化接口供模型开发者与强化学习框架交互，便能实现大规模、稳健、高效且可复现的训练与评估。反过来，这也能让研究社区专注于研究问题，而不必在结合网络仿真与强化学习训练时重复造轮子。值得注意的是，*ns3-gym* [12] 是一个基于 ns-3 的平台，其目标与本文所述工作相似。尽管*ns3-gym* 是朝正确方向迈出的一步

[1]例如，在 [1] 中，作者将智能体的动作计算/监测间隔固定为 20 毫秒，这表明间隔不可能再短太多。[2]一个突出的例外是 Remy，它是最早的基于学习的拥塞控制算法之一[42, 53]。提交给 ACM 的稿件

并且对我们的工作有所补充，但我们也发现了若干局限：由于学习框架与网络仿真器之间需要通过进程间通信（使用*ZeroMQ*）来传递强化学习观测、奖励和动作，因此执行开销较大；在设计层面也存在根本性限制，因为它不支持长度可变、可动态调整的步骤，也不支持多智能体训练/评估（其中各智能体可彼此独立地执行步骤），这两点都严重限制了其在现实强化学习问题中的适用性。

本文介绍*RayNet*，这是一个可扩展且灵活的平台，用于开发基于强化学习的网络协议。RayNet 以优雅且节省资源的方式整合了两种最先进的框架，即 OMNeT++ [50] 和 Ray/RLlib [32], 。OMNeT++ 是一款先进的分组级离散事件网络仿真器，广泛应用于网络研究社区，并支持完全可复现的仿真。Ray 是一个通用的分布式计算框架，旨在灵活地执行任何以 Python 编写的计算密集型任务，包括分布式训练、超参数调优、深度强化学习（DRL）和生产模型服务。尤其是 RLlib [24]，它是一款开源强化学习工具包，通过采用细粒度的嵌套并行，在各种强化学习工作负载上实现最先进的性能，并提供可扩展的抽象，以尽可能减少编程开销来构建新的强化学习算法。RayNet 深度嵌入 OMNeT++ ，直接在模拟器的事件循环中运行，并在需要时控制仿真；也就是说，在模拟网络环境中收集经验——观测和奖励——并更新智能体的动作。同时，RayNet 也作为 Ray 训练器运行，使用户能够通过一组 Python 绑定，在仿真网络上进行分布式多节点训练。这种集成效率很高，唯一的（极小）开销来自 Python 绑定，它使 Ray 能够控制网络仿真。RayNet 通过抽象出仿真中与强化学习相关信息传入和传出的所有实现细节，并向开发者提供简洁的 API，促进代码复用；开发者可以借此将强化学习嵌入模型中，用于训练与评估。我们以一种强化学习驱动的拥塞控制方案作为案例研究，展示 RayNet 如何助力大规模复杂网络问题中强化学习解决方案的设计、工程实现与评估。通过实验，我们展示了 RayNet 如何支持对强化学习驱动的拥塞控制协议进行优化和分析，将学习逻辑配置与网络环境设置解耦，并提供多智能体仿真平台。据我们所知，RayNet 是首个端到端集成 Ray/RLlib 和 OMNeT++的平台，为训练和评估基于强化学习的网络协议提供可扩展的平台，且开销极低。我们将 RayNet 的性能与*ns3-gym* 进行比较，以展示其优越性。RayNet 可作为开源项目在*https://github.com/giacomoni/raynet*获取。

2 背景

本节将简要介绍与 RayNet 相关的工作。首先，我们概述强化学习和拥塞控制。然后，我们讨论 OMNeT++，重点介绍其离散事件特性以及用于控制网络仿真的编程接口；随后，描述 Ray 和 RLlib 的关键特征。最后，我们介绍采用计算机网络模拟器开展强化学习研究的相关文献，并强调综合性开源平台的稀缺性，例如本文提出的平台。

2.1 强化学习

强化学习是通过将状态映射为动作，学习如何最大化数值奖励信号的过程。智能体以随机决策策略初始化，因此必须通过试验来确定哪些动作能带来最高奖励。动作不仅可能影响即时奖励，也可能影响后续的部分或全部奖励。

强化学习问题可以形式化为对部分可观测马尔可夫决策过程（POMDP）的最优控制，这是一种将问题描述为通过交互学习以实现目标的方式。智能体与环境在一系列离散时间步骤中的每一步进行交互，t_0，t_1，t_2，t_3，..... t_n在每个时间步骤 t，智能体都会收到对环境状态 $S_t \in \mathrm{S}$ 的表征，并选择一个动作 $A_t \in \mathrm{A}$。一个时间步骤之后，部分由于自身动作的影响，智能体会收到数值奖励 R_{t+1}，并进入一个新状态 S_{t+1}。马尔可夫决策过程与智能体共同产生一条轨迹，即由状态、动作和奖励组成的序列 S_0，A_0，R_1，S_1，A_1，R_2，S_2，A_2，R_3，... ...。强化学习智能体的目标是找到策略，即状态与动作之间的映射 π：$\mathrm{S} \rightarrow \mathrm{A}$，使折扣累积奖励最大化：

$$\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

其中 $\gamma \in [0, 1]$ 是折扣率。深度强化学习将强化学习方法与非线性函数逼近（即深度神经网络）相结合，以应对这样的目标任务：遇到的状态可能从未出现过。为了在这种情况下做出合理判断，必须根据以往不同条件下的经验进行泛化，而这些条件在某些方面与当前条件具有可比性。关于强化学习的详细讨论见 [46]。

强化学习算法需要采样经验轨迹，以改进当前策略。在线策略算法从策略自身采样的经验中学习，而离策略算法则使用（或重复使用）由其他策略收集的经验来改进当前正在优化的策略。对于某些任务而言，收集经验轨迹并非易事。首先，强化学习采用试错搜索，这涉及探索与利用之间的权衡；为了获得较大的累积奖励，智能体会倾向于采取先前尝试过且已证明能带来奖励的动作。然而，要发现这类动作，它必须尝试此前未选择过的动作。在某些应用中（例如自动驾驶），探索所带来的风险可能很高，因此智能体必须先在安全的环境中训练，然后才能部署到真实环境中。其次，有些任务可能持续很长时间，收集经验样本也可能耗时。因此，使用仿真为强化学习智能体提供高效且安全的训练环境十分常见。随后，智能体可以利用学到的策略部署到“实际场景”中；该策略可能会，也可能不会，根据部署后的智能体收集到的经验进行更新。

2.2 拥塞控制

多个用户访问网络时，必须共享可用资源——带宽和缓冲区。网络拥塞是指一种网络状态，其特征是网络延迟和丢包率上升；这是因为流量经过一个或多个瓶颈链路时，所需带宽超过了可用带宽。网络拥塞会严重降低用户的体验质量，因此必须加以控制。拥塞控制涉及端主机，也可能涉及网络内设备，旨在最大限度地提高资源利用率，同时公平地将资源分配给所有用户。通常通过调节发送端的传输速率，以端到端方式实现这一目标。

计算机网络的去中心化特性，加上应用需求和网络架构的异质性，使拥塞控制本质上成为一个复杂的问题。网络和流量模式不断演变，新的网络架构也不断涌现。这要求我们频繁重新审视拥塞控制算法，或引入新的算法。自传输控制协议（TCP）中引入首个算法以来，已有多种拥塞控制算法在实际网络中部署，例如 TCP Cubic [13],TCP Compound [43] 和多路径 TCP [37]。针对数据中心（[3, 30, 54]）、无线（[21, 28, 41]）和卫星（[2, 47]）网络，研究者提出了大量算法；这些网络的特性和面临的挑战都大相径庭。已部署的拥塞控制协议采用数据包滑动窗口来控制在途数据包的数量，并调整窗口大小 提交给 ACM 的稿件

以应对拥塞事件（例如推断出丢包）。历史上，拥塞控制算法分为两类：基于丢包和基于时延。基于丢包的算法（如 TCP Cubic [13]）根据推断出的丢包情况执行拥塞控制；窗口大小不断增加，直至推断出丢包，此时窗口大小便会减小。基于时延的算法（如 TCP Vegas [6], ）将分组时延增加视为拥塞的信号，并调整窗口以主动降低时延。

近年来，一种新的基于学习的拥塞控制范式逐渐兴起，其核心观点是：拥塞信号和控制动作过于复杂，人类难以理解，而机器生成的算法能够提供优于人工设计算法的策略。随后，通过目标函数引导控制策略的学习。该方向的早期研究包括对固定规则表的离线优化 [42, 53] ，以及在线梯度上升优化 [8, 9], ；之后的研究则采用基于强化学习算法的序贯决策优化[1, 18, 23]。为简洁起见，也为避免重复描述基于强化学习的拥塞控制方法，请参阅第 5 节；我们将在该节中结合 RayNet 实现的用例，详细讨论一种具有代表性的基于强化学习的拥塞控制问题表述。

2.3 OMNeT++模拟器

OMNeT++ 是一种可扩展、基于组件的 C++ 仿真框架，用于开发网络模拟器。这里的“网络”一词含义广泛，涵盖有线和无线通信网络、片上网络、排队网络等。仿真模型由一个或多个组件构成，这些组件封装网络功能（例如 TCP/IP 协议栈、通信信道），并通过门相互交互。OMNeT++ 是一种离散事件仿真器，这意味着时间通过对仿真未来的事件进行调度而向前推进。事件队列（在 OMNeT++中称为未来事件集）是一种关键的数据结构；模拟器按时间顺序执行事件，直到队列中不再有事件，或达到其他终止状态；例如，达到预设的最大时长，或某个特定于模型的终止状态谓词为真，从而以编程方式结束仿真。

OMNeT++ 仿真以单线程进程[3]的形式运行；从较高层面来看，OMNeT++ 仿真的生命周期如下：首先导入并初始化网络模型，并在一组描述性文件（称为 NED 文件）中声明和互连所有仿真组件，这些文件共同构成模型。NED 文件规定仿真模型的结构，例如网络中的节点数量、节点之间的链路，以及每个节点支持的协议栈。配置（.ini）文件设置模型组件的运行方式，包括每个节点上运行的应用程序类型、流量类型和物理链路属性。随后，每个组件都会动态链接到仿真内核。在网络模型初始化期间，可能会实例化一个或多个事件并将其插入事件队列。模拟器程序按照时间顺序处理队列中的每个事件，这可能涉及通过网络链路发送消息，或执行超时事件。处理事件可能会产生新事件；例如，计时器到期后被重置，并重新安排在仿真未来的某个时刻触发。OMNeT++ 导出应用程序编程接口（API），可通过该接口以编程方式执行和控制仿真。借助此 API，程序员可以遍历队列中的事件并逐一处理；如第 4 节所述，这一功能是 RayNet 与 Ray 和 RLlib 集成的关键。

[3]OMNeT++ 支持并行仿真，但本文仅关注单线程仿真。在第7节中，我们将讨论与 RayNet 及并行仿真相关的未来工作。

2.4 Ray 和 RLlib

Ray [32] 是一个通用集群计算框架，支持包括仿真、训练和强化学习应用服务在内的多种功能。RLlib [24] 是一个开源库，提供可扩展的强化学习软件原语，使多种算法能够以高性能、高可扩展性实现，并大量复用代码。RLlib 支持多种用于训练智能体的环境接口。[4] *OpenAI Gym* 是单智能体训练环境的主要接口。回合开始时，系统会向智能体返回初始观测，并将环境重置为其初始状态。智能体通过提供动作与环境交互，并获得该动作及后续观测对应的奖励。此交互以步数为单位进行，直到满足回合终止条件：要么环境达到终止状态，要么达到环境的最大步数。在多智能体环境中，多个智能体可以同时、依次或以两者结合的方式采取动作。*MutiAgentEnv* 接口支持将轨迹映射到各个智能体，并为不同智能体分配不同策略。一个智能体可以映射到单个策略，而一个策略可以映射到多个智能体。

2.5 基于强化学习的研究与仿真

如引言中所述，基于强化学习的网络技术研究层出不穷，其中许多研究都涉及在模拟网络环境中的训练与评估。在此，我们讨论 RayNet 希望填补的空白：提供一个具有标准化接口的仿真平台，使研究人员能够专注于当前问题，而不是采用仅适用于特定研究问题和网络环境的定制方法重复造轮子，因为这可能效率低下且容易出错。我们研究了大量关于拥塞控制、路由、缓存、频谱接入和信号处理系统的基于强化学习的方法，并在下文简要归纳了用于学习智能体策略的仿真方法。[5]

鉴于大多数（如果不是全部）学习和数据处理框架都是使用 Python 实现的，或提供 Python API，大量研究都基于定制的 基于 *Python* 的模拟器。此类基于强化学习的研究包括车载网络中的内容缓存 [38]、物联网 [22, 34, 63],中的路由 [60] 以及智能网络编码 [51]。此类模拟器只专注于特定问题，因此即使公开发布（这属于例外情况而非普遍做法），对其他网络研究子领域也几乎没有用处。另一类常见的定制模拟器是基于 MATLAB 的模拟器。相关示例包括频谱接入 [33], 、内容缓存 [19, 61], 、车载网络中的路由 [59] 以及信号处理 [29]。同样，这些模拟器的适用范围似乎非常狭窄，且无法在不同问题领域和网络环境之间实现复用。

一些用于路由优化 [45, 55, 58] 和缓存 [4] 的基于强化学习的方法，将 OMNeT++ 与某种学习框架（如 TensorFlow）结合起来（例如， [55, 58]）。然而，这些方法都没有像 RayNet 那样将 OMNeT++ 集成到学习框架中。尽管这些论文都没有提供源代码，但显然，OMNeT++ 与学习框架是分开使用的：OMNeT++生成观测和奖励，随后学习框架从外部获取它们（推测是从 OMNeT [55],输出的已存储日志中获取）。例如，++ 的作者指出：“给定路由方案和流量矩阵，OMNeT[58], 仿真环境可以获取网络性能参数（如吞吐量、时延），并将其作为深度确定性策略梯度路由优化机制的奖励。”在++中，作者指出：“给定流量矩阵和路由方案，我们可以直接从 OMNeT [44], 获取网络性能参数，例如网络延迟。”在 中，针对路由优化也介绍了类似的设置，但是

[4]关于 RLlib 支持的环境的精彩讨论，请参见 https://docs.ray.io/en/latest/rllib/rllib-env.html。[5]为简洁起见，我们没有收录所有相关论文，而是从每个类别和研究领域中引用了具有代表性的论文。提交给 ACM 的稿件

其源代码已公开。在该工作中可以看到，OMNeT++ 仿真独立于学习框架运行，强化学习专用数据从模拟器的标准输出中收集，随后输入学习框架。显然，所有这些方法都是针对特定问题和设置定制的，缺乏能够提升框架可复用性和效率的集成，因此无法与 RayNet 直接比较。与上述情况类似，一些用于基于强化学习的路由的方法（[10, 57]）采用 ns-2 和 ns-3 进行仿真；然而，这些方法没有提供源代码，解决方案仅针对路由问题，而且没有迹象表明模拟器与学习框架之间尝试过任何形式的集成，除了在仿真运行结束时收集统计数据并将其输入学习框架。

据我们所知，只有另一种方法与 RayNet 有着类似的目标，那就是 *ns3-gym* [12]。*ns3-gym* 的核心是实现了一个 Python OpenAI Gym 环境，将 ns-3 仿真作为独立进程运行。它使用基于套接字的进程间通信库（*ZeroMQ*），支持 Python 进程与 ns-3 进程之间的通信。这与 RayNet 不同，后者通过使用 *pybind11* 将模拟环境集成在单个进程中，从而实现了高得多的效率。在设计层面，*ns3-gym*中的环境按固定的仿真时间间隔步进，用户必须在 Python 进程运行之前设置这些间隔；这会带来很大的限制，因为并非所有问题都能用固定长度的步骤来建模。重要的是，这也对多智能体训练与评估造成了重大限制。这是因为在*ns3-gym*中，所有智能体都必须同步步进，而现实问题中这种情况很少见。相比之下，RayNet 支持完全可配置的步长，并可在仿真运行期间动态调整，从而支持更复杂的问题设置。例如，在第 5 节中，我们展示了如何使用 RayNet 对拥塞控制问题进行建模：强化学习智能体将网络流作为模型，可以动态加入或退出，并彼此独立地步进。这种建模方式对于研究基于强化学习的拥塞控制非常重要，但使用*ns3-gym*则无法实现。最后，*ns3-gym* 要求智能体从仿真开始时就开始步进；对于智能体只有在特定事件发生后才开始作用于环境的任务而言，这可能会造成问题。拥塞控制就是这种情况：强化学习智能体在慢启动阶段之后才开始步进（例如，如 [1] 所示），这在 *ns3-gym* 中无法实现，但 RayNet 可以轻松支持。

3 设计原则

环境与学习的分离。 与标准的强化学习训练平台一样，环境必须在逻辑上与学习过程分离，其执行需遵循若干简单操作（在程序层面，则需遵循相应的 API）。这样一来，无需更改智能体运行的环境，就可以调整学习算法和超参数。反过来，环境也必须能够改变，而不需要对学习基础设施做任何更改，从而支持在不同情境和场景中进行学习。这种分离的典型示例是*OpenAI Gym* 抽象，它已广泛应用于众多强化学习设置中。在计算机网络领域，这种分离尤为重要，因为单个智能体可能需要在多种网络设置中接受训练，这些设置在物理拓扑、端主机数量和流量工作负载方面各不相同。根据问题的性质，步长的定义可能有所不同。通常，国际象棋或围棋等游戏采用离散的回合制结构，每个步骤对应一名玩家的回合。其他任务，包括 Atari 游戏和机器人控制，则需要对时间进行离散化。智能体按预先设定的时间间隔采取动作，这些间隔标志着一个强化学习步，其长度取决于具体任务。因此，支持对步长进行细粒度控制十分重要；智能体可以彼此独立地推进，也可以分组推进，或者

分别进行。RayNet 通过让 Ray 直接控制每个仿真的事件执行循环，支持基于步进的方法（构建于*OpenAI Gym* 环境之上），详见第 4.2 节。

支持多智能体环境。 学习拥塞控制和路由等核心网络功能的策略，需要在单个环境中运行多个智能体。例如，在学习过程中，多个 TCP 流（即使用相同策略的拥塞控制智能体）可能会竞争网络链路上的带宽。同样，网络中的多个路由器可能会独立处理各个流，并遵循相同或不同的策略。因此，必须支持由多个智能体执行环境（用于训练和/或评估），无论这些智能体是否学习或使用相同的策略。RayNet 通过将 Ray/RLlib 的多智能体接口与定制的信令系统相结合来实现这一点。我们使用 OMNeT++ 的 API 开发了该系统，用于分发和收集动作、奖励和观测，详情见第4.3节。

可复现性。 可复现性使研究人员能够重现已发表的结果、发现错误或局限性，并提出改进方向。由于算法固有方差、环境随机性，以及可能有大量未报告的超参数，强化学习通常难以复现。RayNet 以 OMNeT++ 作为底层环境来收集经验并优化智能体策略，从而尽可能减少导致结果无法复现的因素。OMNeT++ 仿真在设计上具有确定性，因此能够得到完全可复现的结果；OMNeT++ 精密的伪随机数生成器框架则能够控制随机性，并确保同一仿真（即 RayNet环境）的多次运行真正相互独立。此外，Ray 和 RLlib 支持最先进的超参数报告方式，将非确定性限制在算法固有方差之内。**效率与可扩展性**。强化学习需要收集海量经验，智能体借此学习如何与环境进行最佳交互。随后，经验会被存入回放缓冲区，并通过从中抽取经验批次来进行学习。在这一过程中有三个关键问题：（1）环境必须能够快速地将动作转换为奖励和某种部分可观测状态；（2）学习过程本身必须高效；（3）必须尽可能高效地利用所有可用计算资源，包括支持尽可能多的环境并行实例化和执行，并尽量减少强化学习设置中学习组件与环境执行组件交互所产生的开销。RayNet 遵循这一原则，使用轻量级 Python 绑定（见第 4.1 节）以编程方式将 Ray/RLlib 与 OMNeT++ 集成。Ray 支持并行运行多个环境，而 OMNeT++ 本身则将每个环境（即网络仿真）作为单线程进程高效运行，从而实现大规模并行学习。

4 RayNet 架构

本节将详细讨论 RayNet。首先，我们概述其架构，略去实现细节，重点介绍环境分别向模型开发者和训练器提供的接口。接着，我们说明 RLlib 环境如何与 OMNeT++ 的事件循环和仿真模型集成。随后，我们重点介绍在 OMNeT++ 中开发的专用信号系统，以便 Ray/RLlib 能够在用户定义的学习步骤中，高效地向智能体传递动作，并收集其观测和奖励。最后，我们讨论如何将训练好的强化学习智能体部署到仿真网络和现实网络中。

4.1 概述

图 1 展示了 RayNet 的高层视图：左侧是一个训练器（Ray/RLlib）实例，右侧是一个包含环境的轨迹采样工作进程；为便于说明，图中仅展示了一个与 Ray 的训练器交互的工作进程。训练器是运行策略优化器和底层强化学习算法的进程。提交给 ACM 的稿件

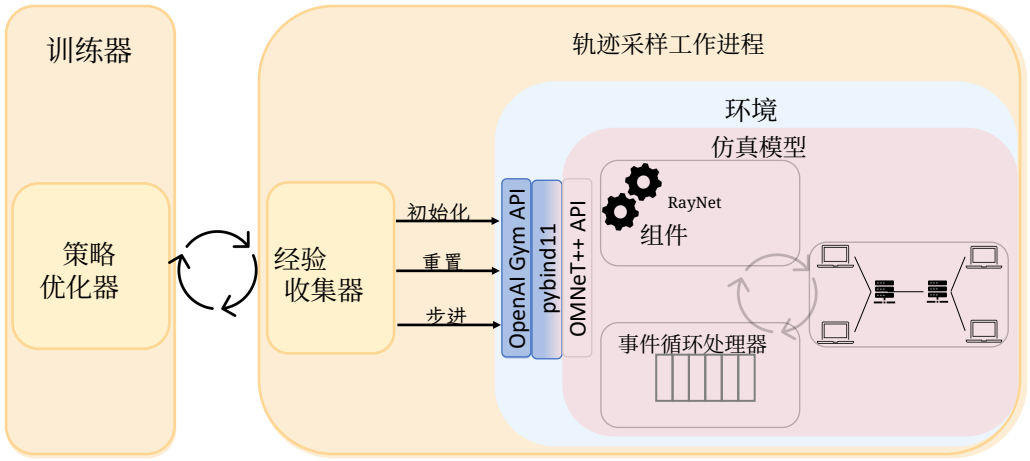

图 1. RayNet 组件概览。训练器和轨迹采样工作进程是 Ray 进程。工作进程会初始化并控制一个实现 OpenAI Gym 接口的 Python 类，该类充当强化学习环境。OMNeT++ 核心库、事件循环处理器、仿真模型和 RayNet 组件通过 *pybind11* API 以编程方式嵌入环境中。

它将策略评估委托给一个或多个轨迹采样工作进程，这些工作进程并行运行，以加快训练期间的经验采集。每个轨迹采样工作进程会被分配一个（如图 1 所示）或多个 RayNet 环境，并通过调用 OpenAI Gym 接口导出的方法与之交互，即 初始化*()*、步进*()* 和 重置*()*，如图 1 所示。环境本身是一个仿真网络，由 OMNeT++ 核心库、事件循环处理器、仿真模型（包括仿真网络和所用的协议栈）以及 RayNet 组件构成。

初始化(): 此方法会创建并初始化环境。所有核心仿真类的对象以及构成网络的类都会被实例化。事件循环处理器也会被实例化，并且作为网络初始化的一部分，会安排一个或多个事件（例如，应用程序向下层数据传输层发送数据包，或无线节点广播链路层帧）。请注意，此时仿真尚未开始；只有调用下文的重置*()* 方法后，回合才会开始。

重置(): 在回合的任意时刻，工作进程都可以调用此方法，将环境恢复到随机状态或起始状态，以便开始新回合。该函数会返回新回合的起始观测。

步骤（动作）： 如第 2.1 节所述，一个回合内的进程通过一系列步骤推进，这些步骤是状态—动作—奖励序列的循环。在 RayNet 中，工作进程通过反复调用此方法来完成这一系列步骤，每次调用都会提供要在环境中执行的动作；随后，环境会转换到下一个强化学习状态。该函数会返回新状态的观测、所执行动作对应的奖励值，以及一个用于指示新状态是否为最终状态（即回合是否结束）的布尔标志。多个强化学习智能体可以同时在环境中执行步骤，此时上述所有标量值都会变为值向量，向量中的每个元素（例如奖励向量）都与一个在环境中运行的特定智能体相关联。步骤、动作、奖励和观测的定义，以及环境如何从一个状态转换到另一个状态，均取决于具体问题，并由仿真模型开发者负责。在训练器中运行的强化学习算法（见图 1）会将奖励和观测输入其内部模型，无需了解环境如何转换；所有特定于环境的知识都封装在仿真模型中。

在仿真环境中，仿真模型可通过下方所示的接口使用强化学习功能。开发者无需关心训练器与模拟器集成的内部机制，也无需处理强化学习智能体与环境之间观测、奖励和动作的传递。这些细节对开发者是隐藏的，并在 RayNet 中实现，具体内容见第 4.2 和 4.3 节。

注册智能体：仿真网络中的每个强化学习智能体都必须向负责处理所有内部通信和步进功能的 RayNet 组件注册（参见第 4.2 和 4.3 节）。此时，环境不会推进新加入的智能体，除非使用下文讨论的方法为其设置步长。因此，智能体加入仿真网络后无需立即开始步进。[6]

设置下一步： 每个强化学习智能体都负责设置自身强化学习步的持续时间；该持续时间可以在整个回合中保持固定，也可以由智能体根据具体问题进行调整。例如，拥塞控制智能体可以将下一步设置为某个估算网络时延的倍数，而该估算值本身会在整个回合中变化。在幕后，RayNet 会安排特殊的步进事件，使仿真环境能够与训练器同步步进，具体讨论见第 4.2 节。请注意，不同的强化学习智能体可能在不同时间步进，并且可以彼此独立地设置各自的步进持续时间。

getObs(): 这是由环境中的每个强化学习智能体实现的回调方法。RayNet 会在一个强化学习步骤结束时调用该方法，它必须生成当前环境状态的观测并将其返回给训练器。观测的计算方式取决于具体问题，训练器本身并不关心观测的语义。**getReward():** 与 *getObs()* 回调类似，RayNet 会调用此回调，通知强化学习智能体必须计算上一步的奖励并将其返回给训练器。其实现方式取决于具体问题。

getDone(): 强化学习智能体必须实现此回调；如果本步骤完成后当前回合已结束，则应返回 true，否则应返回 false。
setAction(): RayNet 会在步骤开始时调用此回调方法，以便强化学习智能体根据训练器维护的强化学习策略更新其动作。与上述回调一样，动作的语义取决于具体问题，只有强化学习智能体才了解。

如第 2.3 节所述，OMNeT++ 导出了一个 API，可通过编程方式初始化并运行仿真；RayNet 利用该 API 控制仿真的生命周期。该生命周期映射到*OpenAIGym* 接口导出的方法，从而有效地将 Ray/RLlib 与 OMNeT++集成起来。我们将在第4.3节详细讨论调用这些方法时 OMNeT++ 仿真内部的交互。集成的最后一部分是使用*pybind11*[7] 实现的 Python 绑定。如图 1 所示，这些绑定位于工作进程与 OMNeT++API 之间，并通过调用直接调用 OMNeT++ API 的 C++ 方法来实现*OpenAI Gym* 方法。

4.2 环境步进与事件循环

在 RayNet 中，智能体与环境之间执行一次完整的交互步骤，需要执行一个或多个仿真事件。执行事件后，仿真时间会推进到该事件原定执行的时间。RayNet 通过直接访问 OMNeT 的事件处理循环，将其集成到 Ray 工作进程中。清单 1 中的伪代码展示了这种集成方式。每当工作进程通过 *pybind11* API（见图 1）调用 *step()* 方法时，控制权就会交给清单 1 中的事件循环。该循环按时间顺序遍历队列中的每个事件并进行处理（第 2–12 行），直到队列中没有更多事件，

[6]如第 5 节所述，这在拥塞控制中十分重要，因为 TCP 发送端必须先经历慢启动阶段，之后智能体才能开始执行步进操作。 [7] https://pybind11.readthedocs.io/ [8]在大规模网络仿真中，单个环境步进可能包含数千个仿真事件。提交给 ACM 的稿件

算法 1环境步进

```
1:  过程步骤
2:      model.setAction(action)
3:      当 queue.size ≠ 0执行
4:          事件 ← queue.get                          ▷ 从事件队列头部取出事件
5:          当前时间 ← 事件.时间戳                    ▷ 推进仿真时间
6:          如果 事件.类型 = 步骤，则
7:              观测 ←模型.获取观测
8:              奖励 ←模型.获取奖励
9:              结束状态 ← model.getDone
10:             返回(观测, 奖励, 结束状态)            ▷ 将控制权交还给工作进程
11:         else
12:             处理(事件)          ▷ 处理过程可能会改变事件队列，或结束仿真/强化学习回合
```

达到模型特定的终止状态，或遇到特殊的步进事件（第 6 行）。在前一种情况下，仿真结束，工作进程会在此方法之外清理仿真。在后一种情况下，则表示一个步骤已结束，工作进程会收集环境中已完成步骤的智能体所获得的奖励和观测（第 7–9 行）；这通过调用上一节讨论的回调方法来完成（即 *getObs()*、*getReward()* 和 *getDone()* ）。我们将在第 4.3 节讨论这些特殊事件如何放入队列。

与所述集成相关的要点有三：(1) 将步进事件放入模拟器的事件队列，实际上使强化学习的步进特性与模拟环境的离散事件特性保持一致。通过要求强化学习智能体调用 *setNextStep()* 方法，显式定义其步骤的持续时间（该方法在幕后会将步进事件安排在模拟未来的某个时刻），强化学习与仿真便能同步推进。(2) 步进细节和环境内部机制均对 Ray/RLlib（即训练器和工作进程）隐藏。工作进程只需调用 *step()* 方法；该方法会依次处理 OMNeT++ 的队列中的事件，从而推进仿真（即执行环境中的操作），而无需了解这些事件的语义。(3) 至关重要的是，步进语义由仿真模型定义，而强化学习智能体（即实现强化学习智能体的开发者）有责任调用 *setNextStep()* 方法，将步进事件放入队列；这使得步进语义可以根据具体问题灵活定义。例如，在拥塞控制问题中，一个步骤可以持续固定时长，也可以持续到发送方收到特定数量的分组确认为止。

4.3 RayNet环境

RayNet 环境由 OMNeT++ 仿真模型（以 C++ 模块的形式实现）以及 RayNet 专用模块组成（即强化学习智能体、步进器 和代理，如图 2 所示）。这些模块用于实现环境步进，并促进环境与 Ray 工作进程之间的交互。RayNet 环境包含一个或多个依据训练好的策略行动的强化学习智能体。例如，在拥塞控制场景中，智能体可以在数据传输层运行，控制特定网络流的传输速率（关于拥塞控制用例的更多详情，请参见第 5 节）。步进器 负责与强化学习智能体协调，使环境能够在 Ray 工作进程调用步进方法时转换到新状态。代理 负责对动作/观测/奖励值（标量或向量）进行序列化和反序列化，并在每个步骤开始和结束时，分别将这些值分发给智能体（动作）和 Ray 工作进程（观测/奖励）。

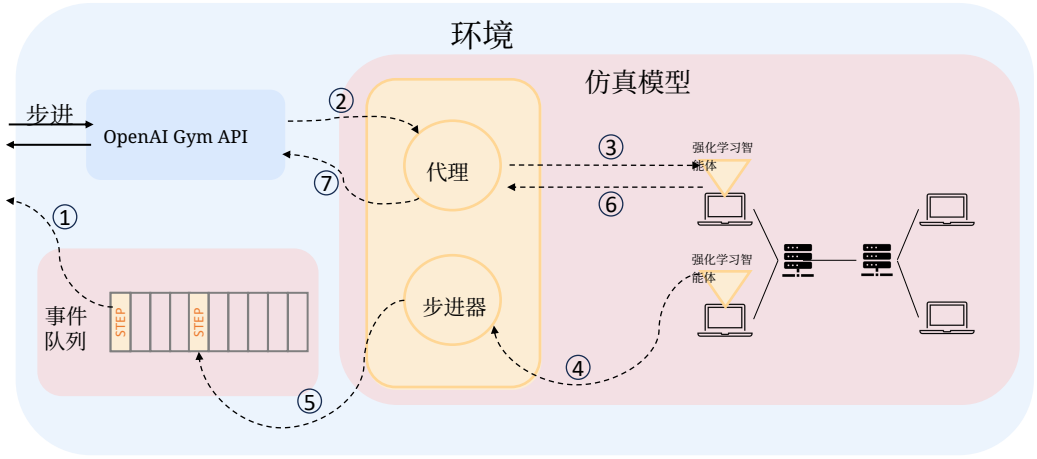

图 2. 一个步骤生命周期内的模块交互。黄色组件属于 RayNet 的实现部分。强化学习智能体以三角形表示，可通过实现第 4.1 节所述的 API 附加到任意仿真模型。

RayNet 专用模块之间的交互通过 OMNeT++提供的信令系统实现，该系统采用发布/订阅通信范式。OMNeT++ 模块可以按名称订阅一种或多种信号类型；多个模块可以订阅同一信号。当某个模块发布其他模块已订阅的信号类型之一时，OMNeT++ 会通知这些模块。这种范式的一个主要优势是发布者模块与订阅者模块之间的耦合较为松散；也就是说，这些模块无需彼此了解即可进行通信。这一点对 RayNet 至关重要，因为环境在其生命周期内可能包含多个在不同时间出现或消失的智能体，例如各个 TCP 流对应的 TCP 发送方。借助 OMNeT++的信令系统，强化学习智能体可以通过发布和订阅预先指定的信号类型，与步进器 和代理通信，而无需在编译时相互引用。

环境初始化时，步进器 和代理 也会随之初始化。在此过程中，它们会订阅特定的信号类型，以便（1）彼此协调；（2）接收强化学习智能体发送的消息。环境初始化时已存在的强化学习智能体会通过发布特定类型的信号，向步进器 和代理 登记其存在，并订阅特定类型的信号，以便接收它们发送的消息；在环境生命周期中出现的强化学习智能体，也会以相同的方式向步进器 和代理登记。此时，初始化已完成，触发上述操作的初始化 方法将控制权交还给 Ray 工作进程。接下来，Ray 工作进程调用重置 方法，使环境进入可以执行第一步的状态。这可能需要处理零个或多个在初始化期间排入队列的仿真事件，以及重置环境时需要安排并执行的事件。步进器 通过将一个步进事件插入队列头部（即，将事件安排在当前仿真时间执行）来发出信号，表示环境已进入该状态。此时环境已准备好步进，重置 方法会将控制权连同初始环境观测一并交还给 Ray 工作进程。随后，Ray 工作进程会反复调用步进 方法，使环境持续步进，直至回合结束——回合的定义取决于具体问题，例如仿真结束或达到某个内部里程碑。环境通过从步进 方法返回并将相应的布尔标志设为 true，来表示回合结束。

图 2 展示了一个强化学习步期间 RayNet 执行的操作顺序。当 Ray 工作进程调用步进 方法时，先前插入事件队列的步进事件会被取出，一个新的强化学习步随即开始（图 2 中的（1））。动作通过 OMNeT++ API 直接传递给代理，该 API 支持按名称访问 OMNeT++ 模块（图 2 中的（2））。根据环境中强化学习智能体的数量，该值可以是标量，也可以是一个值向量（实际上是由 {agent-id, action} 组成的数对），每个强化学习智能体对应一个值。随后，代理 通过发布强化学习智能体已订阅的信号，广播这些动作（图 2 中的（3））。接着，RayNet 会为每个强化学习智能体调用*setAction()* 回调，此时它们都已知晓当前步骤必须执行的动作。如第 4.2 节所述，强化学习智能体必须通过调用*setNextStep()* 方法来设定其步骤的持续时间。调用该方法时，会向步进器 发送一个信号（图 2 中的（4）），随后步进器会在适当的时间为一个或多个强化学习智能体安排步进事件（图 2 中的（5））。一个强化学习步结束时，RayNet 会为该步已结束的每个强化学习智能体调用*getObs()*、*getReward()* 和*getDone()* 回调方法。因此，强化学习智能体会向代理 发送其观测和奖励信号（图 2 中的（6））。最后，当 Ray 工作进程在事件队列中遇到步进事件时，会通过 OMNeT++API 直接从代理 收集观测和奖励（图 2 中的（7））。这些数据随后会传回 Ray 训练器，由其存入强化学习回放缓冲区。

4.4 策略部署

在 RayNet 的模拟环境中部署学习得到的策略非常简单；Ray/RLlib 支持重新加载先前训练好的策略，可用于评估，也可从某个已保存的检查点恢复训练。在评估期间不需要探索，计算动作时可以将其禁用。事实上，除非奖励属于观测的输入特征，否则智能体在决策过程中并不需要奖励，奖励仅用于策略训练。在第 6 节中，我们展示了训练好的智能体在各种环境中的表现，而这些环境全部是模拟环境。在真实网络部署中部署通过 RayNet 学习得到的策略，需要：(1) 导出构成该策略的学习得到的模型，这是 Ray/RLlib 的一项功能；(2) 将该策略与所考虑网络功能的真实世界实现集成。例如，在拥塞控制用例中，可以将学习得到的策略集成到用户空间进程中，该进程通过套接字 API 与修改后的 TCP 协议的内核空间实现进行通信，正如[1]中所做的那样。修改后的 TCP 协议收集观测，并将其传递给用户空间进程；用户空间进程随后考虑接收到的观测，计算出一个动作并反馈给它。

5 使用 RAYNET 学习型拥塞控制

本节介绍 RayNet 的一个应用案例：我们开发了基于深度强化学习的拥塞控制，以展示 RayNet 的功能。如第 2.2 节所述，拥塞控制算法会调整流的在途数据量（即已发送但尚未确认的数据），以最大化整体网络利用率，同时尽可能降低感知延迟，并为各流提供公平的带宽分配。基于强化学习的拥塞控制策略会根据网络发出的某种状态信号（例如，数据包往返时间（往返时间）增加，或收到三个重复确认）来决定采取何种动作（例如，增加在途数据量）。

问题定义。 在我们的应用场景中，网络中每条数据传输流的发送端都部署了一个强化学习智能体。对于每条流，数据传输按步骤进行；每个步骤开始时，强化学习策略会将拥塞窗口大小设为选定值——在一个步骤内，不会因收到确认或任何其他网络内信号而更新拥塞窗口大小。根据路径传播延迟和带宽的不同，一条流的稳态拥塞窗口大小可能相差数个数量级。因此，与 [1], 类似，为了

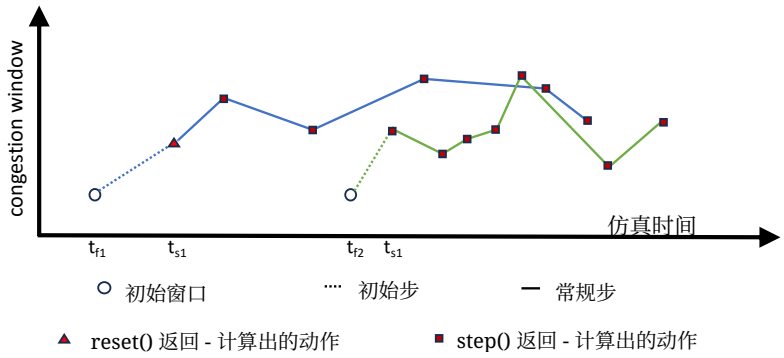

图 3. 包含两个流（智能体）的模拟回合中的拥塞窗口变化。

缩小动作空间，策略动作 α 是应用于当前拥塞窗口值的乘数。在每个步骤 t开始时，强化学习智能体会根据方程 2 设置 $cwnd_t$ 的值。

$$cwnd_t = 2^{\alpha} \times cwnd_{t-1} \tag{2}$$

选择[9] of α 限定在 [-2,2], 范围内，使拥塞窗口最多可增至当前值的四倍，或减至当前值的四分之一。强化学习智能体通过以下观测推断网络状态：

(1) 上一步达到的吞吐量 R 与该流估计的最大带宽 R^{max} 之比。~(2) 上一步测得的最小-最大归一化平滑往返时间 d，其中最小和最大往返时间（RTT）值 d^{min} 和 d^{max} 分别根据流启动时的数据估算。(3) 上一步丢失的数据包 L 与传输的数据包总数之比。(4) 当前拥塞窗口大小。

每个步骤中，分配给智能体的奖励 r 取决于上一步测得的吞吐量 R、往返时间 d 和丢包率 L，具体如下：

$$r = \begin{cases} \frac{\mathcal{R}}{\mathcal{R}^{max}} - L, & \text{if } \frac{\mathcal{R}}{\mathcal{R}^{max}} - L < 1 \wedge d = d^{min}. \\ (\frac{\mathcal{R}}{\mathcal{R}^{max}} - L) \cdot \frac{d^{min}}{d} \cdot (1 - \tilde{d}), & \text{otherwise.} \end{cases} \tag{3}$$

其中为 $\tilde{d} = \frac{d - d^{min}}{d^{max} - d^{min}}$。

一个回合示例。包含两个流（因此包含两个强化学习智能体）的回合生命周期如图 3 所示。环境初始化后，流 1（以蓝色显示）计划在时间 $^{t}f_1$ 首先启动，流 2（以绿色显示）计划在时间 $^{t}f_2$ 启动。调用 *reset()* 方法后，仿真循环开始执行事件，直到第一个流启动。我们采用慢启动阶段（如 TCP 一样），在此期间，拥塞窗口最初设置为较小的固定值（如 [13] 中所示），并在每个往返时间内呈指数增长，直到达到阈值或发生丢包。这样，发送方就能准确估计当前的往返时间、

[9]我们不详细讨论选择特定动作、观测和奖励背后的理由，此处仅为完整起见加以介绍，以便讨论第 6 节中呈现的实验结果。提交给 ACM 的稿件

观测到的最小往返时间和观测到的最大吞吐量；前者用于设置第一步的持续时间，后两者则用于计算奖励。慢启动阶段结束后，该流向步进器发布信号以登记自身，并调用*setNextStep()*声明其初始步骤的持续时间。在时间 t_{s1}，强化学习智能体计算初始步骤的环境观测，并将其发布到代理，随后从*reset()*方法返回。根据接收到的观测推断出下一个动作后，Ray 工作进程调用*step()*方法，将新动作传递给环境。网络中的每个强化学习智能体（每个流对应一个）都独立执行步进，每个步骤持续的仿真时间等于该发送方在过去 10 秒内针对该流观测到的最小往返时间的两倍。每个强化学习智能体通过调用*setNextStep()*方法，向步进器声明自己的步骤持续时间；请注意，如图 3 所示，步骤持续时间可能会随流的生命周期以及网络状况而变化。流 2（以绿色显示）在时间 t_{f2} 启动，此时正处于流 1 的某个步骤中间。流 2 也会执行其初始步骤；在时间 t_{s2}，并且在流 1 执行步进之前，系统返回第一个观测值；接收到根据强化学习策略生成的新动作后，流 2 开始新的步骤。请注意，不同流的强化学习智能体彼此独立地执行步进，收集观测和奖励，并将这些数据全部输入同一个训练过程；也就是说，学习单一的强化学习策略。

```
类 CCAlgorithm: RLInterface {                    // 智能体继承 from 强化学习接口 (参见 章 4.1 节)

  整数拥塞窗口 = 1;                                // 拥塞 窗口 值（以 数据包计）
  模拟时间_t d_分钟 , d_最大值 , d;                // / / 观测到的最小值 / 最大值和当前往返时间值
  双精度浮点数 R, R_最大值, L;                     // 当前且 观测到的最大吞吐量和损失率

  void 收到确认() {                                // 由.....调用 仿真 每个模型 收到确认的时间
    if （慢启动是否完成()）{                       // 检查 if slow 开始 阶段 is 已完成
      如果 (!代理是否已注册 ()){                   // 检查 if 智能体 is 注册的
        注册智能体 ();                             // 注 册 智能体 with RayNet
        setNextStep (2 * 延迟_最小);               // 定 义 长度 的 第一 步骤
      }

      更新网络变量 ();                             // 更新 延迟_min 延迟_max d, R, 最大半径和损失率_

    } 否则 {
      执行慢启动 ();                               // 拥塞窗口 = 拥塞窗口 + 1;
      更新网络变量 ();                             // 更新 延迟_最小值 延迟_最大值 d, R, 最大半径和损失率_
    }
  }

  // 在末尾 of step （长度 定义于 lin^ 11 or 32），RayNet 调用 the 方法 下方

  观测类型 获取观测 () { ... }                     // return 观察 定义 in the list 以上
  奖励类型 获取奖励 () { ... }                     // 计 算 奖励 as per Eq. 3
  bool 获取结束状态 () { ... }                     // 若损失率 >= threshold 为真 or 所有 已确认的数据包

  / / 之 前 的 开始 一个新步骤时, RayNet 调用以下 方法

  void setAction (动作类型 action) {               // 由.....调用 RayNet 就在 在新步骤开始之前 开始
    拥塞窗口 = (2 ^ 动作) * 拥塞窗口               // 应用 动 作 to 智能体的拥塞 窗口 as 根据方程。2
    设置下一步（2 * d_min）;                       // 定义 长度 的 next 步 骤
  }
}
```

清单 1. 基于强化学习的拥塞控制程序摘录

在 RayNet 中编写基于强化学习的拥塞控制程序。 下面简要介绍如何在 RayNet 中实现基于强化学习的拥塞控制用例。为清晰起见，我们略去许多实现细节，重点介绍 RayNet 的核心方法，以及这些方法与发送方（和强化学习智能体）处理确认的交互方式。清单 1 中的编程摘录对此进行了说明。拥塞窗口是由智能体设置的控制变量，定义在第 3 行。连接生命周期内测量的所有变量都定义在第 4 行和第 5 行。这些变量用于计算观测和奖励，如上文所述。*receivedAck()*（第 7 行）由 OMNeT++ 在发送方每次收到确认时调用；拥塞控制的核心就在这里实现。如上文所述，在我们的示例中，慢启动阶段结束后，智能体才会被激活。在慢启动期间，每收到一个确认，窗口就增加一个数据包，从而每经过一个往返时间，其大小便会翻倍（第 17 行）。随后，智能体会注册到 RayNet（第 10 行），并设置第一步的长度（第 11 行）。RayNet 在后台推进环境，具体如第 4.2 节和第 4.3 节所述。第一步结束前，RayNet 会调用第 24、25、26 行中定义的三个回调方法，并将返回值传递给训练器。下一步开始时，会调用第 30 行的*setAction()* 方法，并据此设置拥塞窗口（第 31 行）。同时还会设置步骤长度（第 32 行）。之后每一步都会重复相同的调用顺序，直至回合结束，也就是 *getDone()* 返回 TRUE 时为止；在我们的示例中，这表示所有数据包均已成功传输，或者丢包率达到预先设定的阈值。

6 使用 RayNet 进行实验

本节通过对以下两项进行实验，探讨 RayNet 的功能和性能特征：(1) 第 5 节讨论的拥塞控制用例；(2) 我们也在 RayNet 中开发的简单 CartPole 环境 [5] 。我们的目标是展示 RayNet 满足第 3 节提出的设计原则。[10] 首先，我们展示 RayNet 的设计如何促进复杂网络任务中的学习，该任务涉及搜索和优化强化学习算法及其相应的超参数、在多种网络环境中进行评估，以及分析多智能体性能。其次，我们提供证据，表明在使用不同数量的 CPU 核心执行相同的基本任务（即 CartPole 任务）时，与我们选定的基准（Open AI Gym）相比，RayNet 的计算开销极小。此外，我们还展示了在相同设置下，RayNet 的性能优于 *ns3-gym* 。最后，我们展示了 RayNet 在分布式计算环境中运行拥塞控制任务。实验在一台配备 64 个 CPU 和 128GB 内存的 Linux 服务器以及 Amazon AWS 上进行。

6.1 环境与学习的分离

在此，我们使用第 5 节中讨论的奖励函数和观测来训练模型，以获得高效的拥塞控制策略。具体而言，我们在图4所示的哑铃网络上训练单个强化学习智能体（发送方）。我们对拥塞控制策略采用“训练与部署”方法。

改变环境参数。我们展示了 RayNet 如何在完全独立于学习组件的情况下改变环境。我们让拥塞控制智能体接触多种网络配置，其中网络参数从表1所示的范围内采样。我们使用十六个并行轨迹采样工作进程训练智能体，共进行 100 万步。每个工作进程都会创建自己的强化学习环境，并在表1列出的各个参数范围内均匀采样参数。通过从多个网络场景并行收集经验，可以防止模型过拟合，并避免遗忘训练过程中先前收集过经验的网络场景

[10]我们在此不再进一步讨论可复现性 原则；OMNeT++ 仿真具有确定性，因此任何可复现性限制都仅源于强化学习算法本身的固有特性。提交给 ACM 的稿件

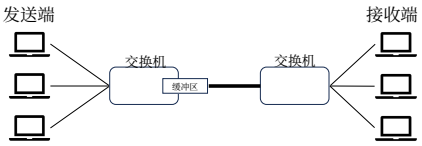

图 4. 哑铃型拓扑。

带宽	RTT	缓冲区
64-128兆比特每秒	16-64毫秒	80-800 个数据包

表1。训练期间网络参数的取值范围

。 [1]我们使用 OMNeT++ 的 NED语言定义参数化网络，并在每个回合开始时设置瓶颈带宽、传播时延和缓冲区大小的值。智能体通过深度确定性策略梯度（DDPG）进行训练。这是一种基于确定性策略梯度、无模型的演员-评论家算法，可以在连续动作空间中运行 [25], ，并结合分布式优先经验回放 [16]。DDPG 的超参数采用 RLlib 的默认值

我们在参数取值范围比训练时更广的网络上评估经深度确定性策略梯度训练的策略，以评估该策略对未见过的环境的泛化能力。评估模型在训练期间未观测到的状态空间区域中的性能，有助于避免部署失败。我们通过在包含且宽于相应训练范围的取值范围内改变所研究的三个维度之一（带宽、传播时延和缓冲区大小），同时将另外两个维度固定为表1所示的相应训练范围的平均值，来评估智能体（见表1）。我们通过测量三个关键性能指标来评估拥塞控制策略：归一化吞吐量（实测吞吐量与理论最大吞吐量之比）、网络瓶颈处的排队时延，以及丢包率。

图5、图6和图7分别展示了带宽、传播时延和缓冲区大小变化时单流的选定性能指标。曲线和蓝色阴影区域分别表示平均值和标准差。横轴和纵轴分别表示变化的参数和测得的性能指标。红色阴影区域表示变化的网络参数的训练范围。尽管分析这些结果超出了本文的讨论范围，为求完整，我们仍简要讨论所学习策略的行为，并指出正是RayNet将环境与学习组件清晰分离，才使得此类分析成为可能。

与传播时延和缓冲区大小相比，瓶颈带宽对吞吐量、排队时延和丢包率的影响最大。事实上，如果网络带宽处于训练范围内，流就能实现最高吞吐量（图 5.a）、最低排队时延（图 5.b）和可忽略不计的丢包率（图 5.c）。如果带宽超出训练范围，当可用带宽低于经验值时，吞吐量会下降（图 5.a）；这是因为较大的拥塞窗口会导致缓冲区过载，从而增加排队时延（图 5.b）和丢包（图 5.c）。当可用带宽高于经验范围时，我们会观察到瓶颈链路利用不足，其特征是吞吐量较低（图 5.a）且没有排队时延（图 5.b）。即使可用带宽未得到充分利用，流仍会发生丢包（图 5.c）。这是因为在带宽时延积（BDP）较大的流中，慢启动阶段的丢包会增加。提高传播时延时，产生的行为与此类似（图 6.c）。将传播时延设在训练范围之外，对性能的影响不如带宽变化那么大。传播时延增加时，慢启动结束时的 BDP 和丢包率都会升高（图 6.c），队列积压也更容易出现（图 6.b）。缓冲区大小的变化主要会增加排队时延（图 7.b）。

变化的学习参数和超参数。 发现并优化强化学习策略通常需要进行实证评估，以确定最佳的强化学习算法和超参数。例如，某些任务本身可能

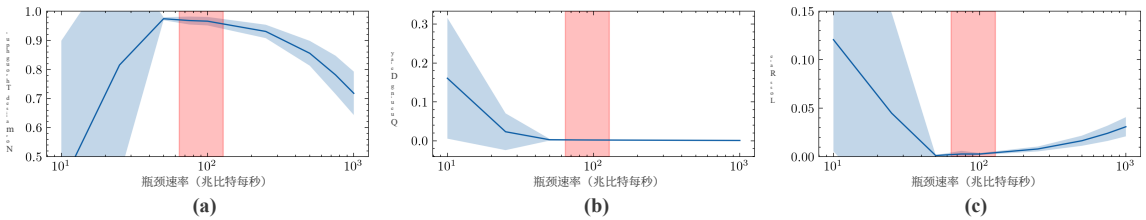

图 5. 单流的归一化吞吐量、排队时延和平均丢包率随瓶颈带宽的变化。阴影区域表示训练期间使用的瓶颈带宽范围。

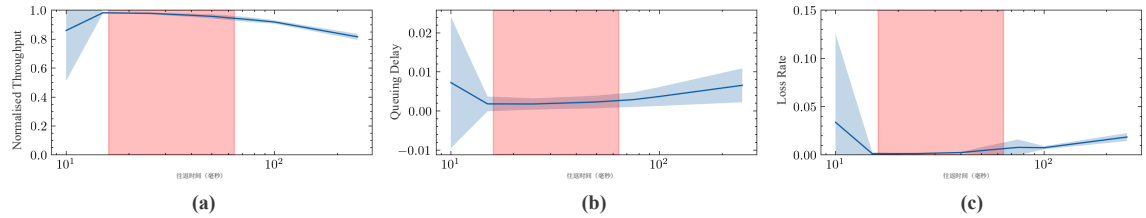

图 6. 单流的归一化吞吐量、排队时延和平均丢包率随传播时延的变化。阴影区域表示训练期间使用的传播时延范围。

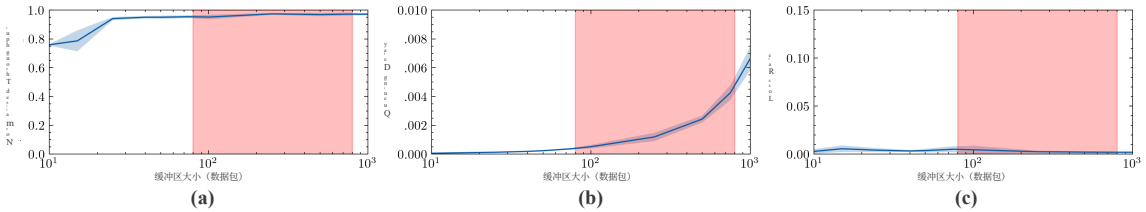

图 7. 归一化　单流的吞吐量、排队时延和平均丢包率随缓冲区大小的变化。阴影区域表示训练期间使用的缓冲区大小范

要求采用随机决策策略来最大化目标，例如石头剪刀布示例，在该示例中，确定性策略必然会导致次优决策。即使确定了强化学习算法，也需要在与所选算法相关的广泛超参数范围内评估其有效性。例如，深度确定性策略梯度需要对探索策略进行精细调节 [25]；软演员-评论家通过在奖励函数形式中最大化策略的熵来促进探索，但代价是引入一个新的温度超参数 [14]，用于权衡探索与奖励；近端策略优化使用替代损失函数，将从旧策略到新策略的步骤限制在安全范围内，这需要使用裁剪阈值、加权 Kullback-Leibler（KL）散度因子，或二者的组合 [40]。多个强化学习算法共用若干超参数，但针对每种算法和具体任务，仍需分别进行优化。例如，离策略算法通常会将经验存储在经验回放缓冲区中，其大小必须预先设定；折扣因子 γ 是大多数强化学习算法实现中的常见超参数；神经网络等函数逼近器 提交给 ACM 的稿件

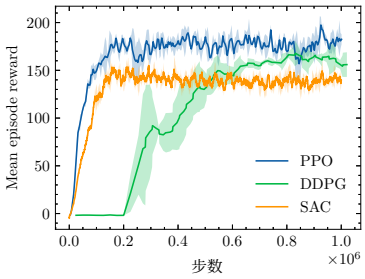

图 8。 训练过程中累积回合奖励随步数的变化。

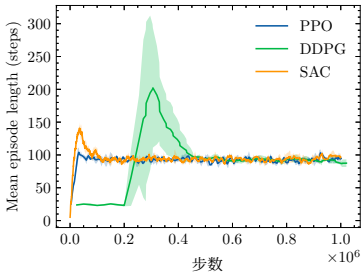

图 9。 训练过程中回合长度随步数的变化。

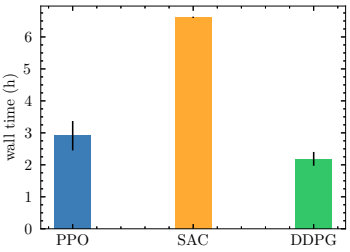

图 10。 使用近端策略优化、软演员-评论家和深度确定性策略梯度进行 100 万步训练的持续时间。

网络带来了整套超参数，例如学习率、损失优化器、激活函数等。RayNet 开箱即支持此类探索性研究，它将 Ray/RLlib 与 OMNeT++ 集成，使强化学习环境与学习组件完全分离。为展示这一能力，我们在上述实验设置中使用三种最先进算法训练强化学习策略：PPO [40], ，一种策略梯度算法；DDPG [25], ，一种结合分布式优先经验回放的确定性策略梯度算法；以及 SAC [14], ，一种演员-评论家算法的软策略优化版本。三种算法都属于 RLlib [24] ，所有相关配置均通过 RLlib 的应用程序接口完成，完全独立于底层环境（这些环境按上一节所述进行参数化）。

图 8 展示了拥塞控制智能体训练期间各回合获得的平均累积奖励。PPO 和 SAC 都优化随机策略，即根据状态生成动作分布；每次策略更新都会受到约束：PPO 通过为策略更新选取安全区域来约束，SAC 则通过施加最大熵原理来约束。在这两种情况下，累积奖励都单调递增，并且两种算法经过相同的训练步数（约 125K）后都收敛至渐近最优值。SAC 的渐近累积奖励取决于熵权重因子，默认的熵最大化策略带来的累积奖励低于 PPO。与此同时，DDPG 优化的是确定性策略。策略更新不受限制，策略变化取决于探索策略。从图 8 可以看出，与 PPO 和 SAC 相比，DDPG 的学习速度较慢，因为它需要先经过 200K 步的预热时间，在此期间随机生成经验。策略开始训练后，累积奖励先达到局部最大值，随后回升并收敛至更高的渐近最大值。这一现象是回合时长不同所致。根据我们对拥塞控制模型的定义，一个回合可以通过以下三种方式结束：(1) 策略产生极高程度的拥塞，表现为推断损失很高；(2) 流成功完成；或 (3) 达到最大步数（本例中为 400 步）。在第二种情况下，累积奖励可能无法准确反映策略本身的质量。从图 8 和图 9 可以看出，较长的回合可能带来更高的累积奖励；但对于大小固定的流，较短的回合意味着更短的流完成时间，这是表现良好的拥塞控制策略所带来的结果。在三类算法中，SAC 完成训练所需的实际运行时间最长，是 PPO 和 DDPG 的两倍（图 10）。

6.2 对多智能体环境的支持

本节通过实验展示 RayNet 对多智能体环境的支持：在图4所示的网络中，两个流在共享瓶颈处竞争资源。我们在一个网络中运行这两个流，该网络具有一个 Manuscript submittedtoACM

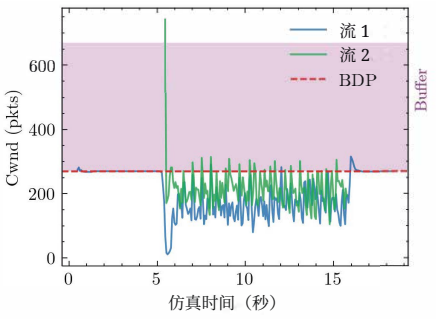

图 11. 使用相同策略的两个竞争流（智能体）的拥塞窗口变化。

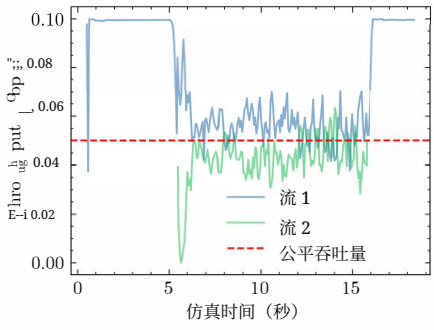

图 12. 使用相同策略的两个竞争流（智能体）所达到的吞吐量。

440 个数据包的缓冲区、一个 100 兆比特每秒的瓶颈以及 35 毫秒的传播时延。图 11 展示了两个流的拥塞窗口大小的变化情况。当第一个流（蓝线）是网络中的唯一流时，其强化学习智能体会调节拥塞窗口，使其与路径的 BDP 相匹配，即在空网络路径中传输的单流所对应的最优窗口大小 [20]。拥塞窗口若再增大，数据就会在瓶颈缓冲区中排队，从而增加该流经历的延迟。图 11 中，模拟开始后的前 5 秒内，蓝线与红线重合，体现了这一点；紫色阴影区域表示会导致数据缓冲的拥塞控制值。当第二个流（绿线）开始传输时，其窗口会在慢启动阶段呈指数增长，直到发生丢包。随后由强化学习智能体接管控制，并调整拥塞窗口，使两个拥塞窗口的总值大致与路径的 BDP 相匹配。在图 12 中可以看到，尽管训练过程中没有经历多个流，该策略仍能在两个流之间相对公平地分配带宽。第二个流结束后，第一个流会迅速增大拥塞窗口，以获取所有可用带宽。

6.3 效率与可扩展性

迄今为止，我们已经展示了 RayNet 如何通过将环境与学习过程分离，并在学习和评估强化学习策略时支持多个智能体，实现对基于强化学习的网络协议进行丰富的实验。重要的是，这些功能不能以学习速度缓慢、资源消耗过大为代价。本节将展示 RayNet 如何利用其固有的并行化能力，以及在 Ray 与 OMNeT++ 之间传递环境观测、奖励和动作时极低的开销，从而高效地学习强化学习策略。为此，我们进行了两项不同的实验：（1）我们在 RayNet 和 *ns3-gym* 中分别将 CartPole 任务*CartPole-v1* 实现为 OMNeT++ 模型，并将这两个模型的学习效率与基于 Python 的 Open AI Gym 中实现的 CartPole 任务进行比较。请注意，我们的目的并非展示 RayNet 或*ns3-gym* 相较于 Open AI Gym 的优势，因为后者的用途截然不同。相反，我们实现了尽可能简单的模型（即 CartPole），以尽可能排除仿真模型本身的计算开销，并将 Open AI Gym 作为性能基准。（2）我们在 Amazon AWS 上部署 RayNet，并在分布式计算环境中测量其学习效率；也就是说，在训练底层强化学习模型时改变所用虚拟机的数量。

在单台服务器上使用 CartPole 任务进行学习。 在 CartPole 任务中，一个不可驱动的连接点将一根杆连接到沿无摩擦轨道行驶的小车上。杆垂直放置在小车上，目标是通过向小车左右两侧施加力来保持其平衡。提交给 ACM 的稿件

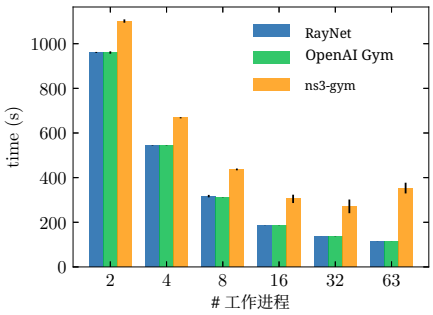

图 13。 在单台服务器上运行数量可变的并行工作进程，以 100K 步训练 CartPole 任务的深度 Q 网络策略时的实际运行时间。

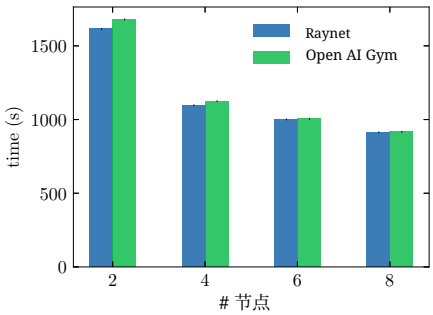

图 14。 在分布式 Ray集群（运行于 Amazon AWS）中，使用数量可变的节点训练 CartPole 任务的深度 Q 网络策略 500k 步时的实际运行时间。

CartPole [7] 状态转移的动力学分别在 RayNet 和 *ns3-gym* 的单个 OMNeT++ 和 ns-3 仿真组件中实现。在 RayNet 模型中，环境初始化期间，代理、步进器 和 CartPole 模块会注册特定的信号类型，以便交换动作、观测和奖励。当对环境调用重置 时，CartPole 组件的内部状态会被设为状态空间范围内的随机值。CartPole 组件会立即将随机生成的观测发送给代理，而步进器 会将一个步进事件插入队列。轨迹采样工作进程从代理 获取观测并计算出新的动作后，会对环境调用步进 ，指示强化学习智能体应用新计算出的动作。代理 通过信号将动作传递给 CartPole 组件，从而触发状态转移和奖励计算。新的观测和奖励会被推送到代理；步进器 会将新的步进事件插入队列，随后步进 方法完成。我们已在*ns3-gym*中将 CartPole 实现为单个组件，并遵循其 API [12]。

对于这三个模型，我们使用数量可变（介于 2 和 63 之间）的轨迹采样工作进程来训练深度Q网络（DQN） [31] 策略。这些工作进程在单台服务器上并行运行，并分别固定在一个可用的 CPU 核心上。训练器本身始终运行在单独的 CPU 核心上。对于每个 <环境和工作进程数> 组合，我们使用三个不同的随机种子训练智能体，并在收集到 100K 步后终止每次运行。在图 13 中，我们观察到 RayNet 的表现与基线 Open AI Gym 实现完全相同，即引入 Ray/RLlib 和 OMNeT++ 所产生的开销可以忽略不计。这一结论适用于我们测试的所有工作进程数量。其次，*ns3-gym* 始终需要更多时间（随着工作进程数量增加，所需时间超过 2× ）才能从相同的环境中收集相同数量的经验。我们只能将其归因于 ns-3 仿真与由 Ray 的轨迹采样工作进程管理的独立 Python进程之间所需的进程间通信。在 *ns3-gym* 中，训练器、轨迹采样工作进程与嵌入式网络模拟器之间需要通过进程间通信来传递动作、奖励和观测。

在 Amazon AWS 上使用 CartPole 任务进行学习。现在，我们使用 RayNet 和 Open AI Gym 的实现，在运行于 Amazon AWS 的分布式集群上为 CartPole 任务训练策略。该集群由多个节点组成，每个节点配备 8 个虚拟 CPU 和 16GB 内存。我们使用深度 Q 网络训练策略，共训练 500K 步数；轨迹采样工作进程的数量设为集群中的 CPU 总数减一，因为与上文讨论的单服务器配置一样，有一个 CPU 留给训练器进程使用。在此实验中，无法纳入*ns3-gym* ，因为它不支持在分布式集群上运行。如图 14 所示，RayNet 的表现与 Open AI 一样出色

Gym，随着分配给任务的底层服务器（和虚拟 CPU）数量增加，学习速度也随之加快。这表明 RayNet 可以为计算机网络问题的强化学习策略训练提供可扩展的平台。

7 结论

本文介绍了 RayNet，这是一种用于训练和评估基于强化学习的网络协议的仿真平台。RayNet 集成了 OMNeT++（一种广泛使用的现成离散事件仿真器）以及 Ray/RLlib（一种支持大规模强化学习的分布式框架）。该集成通过使用 Python 绑定以及 OMNeT++ 导出的信令系统实现，使强化学习智能体能够控制模拟环境中的决策过程。RayNet 致力于成为一个仿真平台，涵盖计算机网络领域中多样化的基于强化学习的研究问题和需求。为此，它抽象 away 了与仿真内外强化学习相关信息通信有关的所有实现细节，并向模型开发者提供简单的 API，使其能够将强化学习嵌入自己的模型中。我们的结果表明，RayNet 的设计有助于基于学习的协议开发；它支持对学习算法和网络环境进行独立且可扩展的配置；支持多智能体模拟，其中每个智能体都可独立执行步骤；与现有的通用强化学习训练平台相比，它引入的额外开销极小，并且性能优于*ns3-gym*，后者与 RayNet 具有相似的目标。

我们未来的工作重点是为 RayNet 开发更多应用场景，包括基于强化学习的路由和流量工程，并提升代码库的代码复用性和易用性。RayNet 作为开源项目向研究社区开放，供其使用和进一步开发。目前，我们正在使用 RayNet 研究公平的拥塞控制算法，并深入研究现有的基于强化学习的拥塞控制模型。未来工作的另一个方向是探究在并行 OMNeT++仿真中使用 RayNet 的优势；在这些仿真中，智能体分布在多个仿真分区中，而这些分区运行于不同的 CPU 核心或服务器上。

参考文献

[1] Soheil Abbasloo、Chen-Yu Yen 和 H Jonathan Chao。2020年。经典与现代的结合：一种务实的基于学习的互联网拥塞控制方法。载于 *ACM SIGCOMM*会议论文集。632–647。[2] Ian F Akyildiz、Giacomo Morabito 和 Sergio Palazzo。2001年。TCP-Peach：一种面向卫星 IP 网络的新型拥塞控制方案。*IEEE/ACM* 网络汇刊 9，第3期（2001年），307–321。[3] Mohammad Alizadeh 等。2010年。数据中心 TCP（DCTCP）。载于 *ACM SIGCOMM*会议论文集。63–74。[4] Nyothiri Aung 等。2023年。VeSoNet：使用深度强化学习的车载社交网络流量感知内容缓存。*IEEE*智能交通系统汇刊 24，第8期（2023年），8638–8649。[5] Andrew G Barto、Richard S Sutton 和 Charles W Anderson。1983年。能够解决困难学习控制问题的类神经元自适应元件。*IEEE*系统、人类与控制论汇刊 5（1983年），834–846。[6] Lawrence S Brakmo、Sean W O'Malley 和 Larry L Peterson。1994年。TCP Vegas：用于拥塞检测与避免的新技术。载于 *ACM SIGCOMM*会议论文集。24–35。[7] Greg Brockman 等。2016年。OpenAI Gym。arXiv：arXiv:1606.01540[8] Mo Dong、Qingxi Li、Doron Zarchy、P Brighten Godfrey 和 Michael Schapira。2015年。PCC：重新架构拥塞控制以实现稳定的高性能。载于 *USENIX NSDI*会议论文集。395–408。[9] Mo Dong、Tong Meng、Doron Zarchy、Engin Arslan、Yossi Gilad、Brighten Godfrey 和 Michael Schapira。2018年。PCC Vivace：在线学习拥塞控制。载于 *USENIX NSDI*会议论文集。343–356。[10] J. Dowling、E. Curran、R. Cunningham 和 V. Cahill。2005年。利用协作式强化学习中的反馈来自适应地优化移动自组网路由。*IEEE*系统、人类与控制论汇刊——*A*部分：系统与人类 35，第3期（2005年），360–372。[11] Benjamin Fuhrer、Yuval Shpigelman、Chen Tessler、Shie Mannor、Gal Chechik、Eitan Zahavi 和 Gal Dalal。2022年。在 NVIDIA 网卡中实现强化学习数据中心拥塞控制。*arXiv*预印本 *arXiv:2207.02295*（2022年）。[12] Piotr Gawłowicz 和 Anatolij Zubow。2019年。ns-3 与 OpenAI Gym 的结合：网络研究中机器学习的试验场。载于 *ACM MSWIM*会议论文集。[13] Sangtae Ha、Injong Rhee 和 Lisong Xu。2008年。CUBIC：一种新的 TCP 友好的高速 TCP 变体。*ACM SIGOPS*操作系统评论 42，第5期（2008年），64–74。M

[14] Tuomas Haarnoja、Aurick Zhou、Pieter Abbeel 和 Sergey Levine。2018年。软演员-评论家：具有随机演员的离策略最大熵深度强化学习。载于ICML会议论文集。1861–1870。[15] Ying He、Nan Zhao 和 Hongxi Yin。2017年。面向联网车辆的网络、缓存与计算一体化：一种深度强化学习方法。IEEE车辆技术汇刊 67，第1期（2017年），44–55。[16] Dan Horgan 等。2018年。分布式优先经验回放。arXiv预印本 arXiv:1803.00933（2018年）。[17] Tianchi Huang、Rui-Xiao Zhang、Chao Zhou 和 Lifeng Sun。2018年。QARC：基于深度强化学习的实时视频流视频质量感知速率控制。载于ACM Multimedia会议论文集。1208–1216。[18] Nathan Jay、Noga Rotman、Brighten Godfrey、Michael Schapira 和 Aviv Tamar。2019年。从深度强化学习视角看互联网拥塞控制。载于ICML会议论文集。3050–3059。[19] Wei Jiang、Gang Feng、Shuang Qin、Tak Shing Peter Yum 和 Guohong Cao。2019年。面向移动D2D网络高效内容缓存的多智能体强化学习。IEEE无线通信汇刊 18，第3期（2019年），1610–1622。[20] Leonard Kleinrock。2018年。使用功率指标进行互联网拥塞控制：让管道刚好满，但不要更满。Ad hoc networks 80（2018年），142–157。[21] Dzmitry Kliazovich、Fabrizio Granelli 和 Daniele Miorandi。2006年。TCP Westwood+ 在高速长距离网络中的增强。载于IEEE ICC会议论文集。710–715。[22] Dohyun Kwon、Joongheon Kim、David A. Mohaisen 和 Wonjun Lee。2020年。面向毫米波车联网视频缓存的深度强化学习自适应功率控制。Journal of Communications and Networks 22，第4期（2020年），326–337。[23] Dehao Lan、Xiaobin Tan、Jinyang Lv、Yang Jin 和 Jian Yang。2019年。一种基于深度强化学习的 NDN 拥塞控制机制。载于IEEE ICC会议论文集。1–7。[24] Eric Liang 等。2018年。RLlib：分布式强化学习的抽象。载于ICML会议论文集。3053–3062。[25] Timothy P Lillicrap 等。2015年。利用深度强化学习进行连续控制。arXiv预印本 arXiv:1509.02971（2015年）。[26] Zoubir Mammeri。2019年。基于强化学习的网络路由：方法综述与分类。IEEE Access 7（2019年），55916–55950。[27] Hongzi Mao、Ravi Netravali 和 Mohammad Alizadeh。2017年。使用 pensieve 的神经自适应视频流。载于ACM SIGCOMM会议论文集。197–210。[28] Saverio Mascolo、Claudio Casetti、Mario Gerla、Medy Y Sanadidi 和 Ren Wang。2001年。TCP Westwood：用于增强无线链路传输的带宽估计。载于ACM MobiCom会议论文集。287–297。[29] Nicholas Mastronarde 和 Mihaela van der Schaar。2011年。用于节能无线通信的快速强化学习。IEEE信号处理汇刊 59，第12期（2011年），6262–6266。[30] Radhika Mittal 等。2015年。TIMELY：面向数据中心的基于RTT的拥塞控制。ACM SIGCOMM计算机通信评论 45，第4期（2015年），537–550。[31] Volodymyr Mnih 等。2013年。利用深度强化学习玩 Atari。arXiv预印本 arXiv:1312.5602（2013年）。[32] Philipp Moritz 等。2018年。Ray：面向新兴人工智能应用的分布式框架。载于USENIX OSDI会议论文集。561–577。[33] Oshri Naparstek 和 Kobi Cohen。2018年。用于分布式动态频谱接入的深度多用户强化学习。IEEE无线通信汇刊 18，第1期（2018年），310–323。[34] Ali Nasehzadeh 和 Ping Wang。2020年。一种面向物联网的深度强化学习缓存策略。载于IEEE/CIC ICCC会议论文集。969–974。[35] Ravi Netravali 等。2015年。Mahimahi：面向 HTTP 的精确记录与重放。载于USENIX ATC会议论文集。417–429。[36] Thanh Thi Nguyen 和 Vijay Janapa Reddi。2019年。用于网络安全的深度强化学习。IEEE神经网络与学习系统汇刊（2019年）。[37] Christoph Paasch 和 Olivier Bonaventure。2014年。多路径 TCP。Commun. ACM 57，第4期（2014年），51–57。[38] Guanhua Qiao、Supeng Leng、Sabita Maharjan、Yan Zhang 和 Nirwan Ansari。2020年。用于车载边缘计算和网络中协作内容缓存的深度强化学习。IEEE物联网期刊 7，第1期（2020年），247–257。[39] Alessio Sacco、Matteo Flocco、Flavio Esposito 和 Guido Marchetto。2021年。Owl：通过强化学习在部分不可见网络中进行拥塞控制。载于IEEE INFOCOM会议论文集。1–10。[40] John Schulman、Filip Wolski、Prafulla Dhariwal、Alec Radford 和 Oleg Klimov。2017年。近端策略优化算法。arXiv预印本 arXiv:1707.06347（2017年）。[41] Hideyuki Shimonishi、MY Sanadidi 和 Mario Gerla。2005年。改善有线无线融合网络中 TCP 的效率与友好性权衡。载于2005年IEEE国际通信会议，ICC 2005。2005年，第5卷。IEEE，3548–3552。[42] Anirudh Sivaraman、Keith Winstein、Pratiksha Thaker 和 Hari Balakrishnan。2014年。拥塞控制可学习性的实验研究。ACM SIGCOMM计算机通信评论 44，第4期（2014年），479–490。[43] Kun Tan、Jingmin Song、Qian Zhang 和 Murari Sridharan。2006年。复合TCP：一种面向高速网络的可扩展且 TCP友好的拥塞控制。PFLDnet 2006会议论文集（2006年）。[44] Giorgio Stampa、Marta Arias、David Sánchez-Charles、Victor Muntés-Mulero 和 Albert Cabellos。2017年。一种用于软件定义网络路由优化的深度强化学习方法。arXiv预印本 arXiv:1709.07080（2017年）。

[45] Penghao Sun、Junfei Li、Zehua Guo、Yang Xu、Julong Lan 和 Yuxiang Hu。2019年。SINET：利用深度强化学习在部分节点上实现可扩展的网络路由。发表于*ACM SIGCOMM* 会议论文集（海报与演示）。88–89。[46] Richard S Sutton 和 Andrew G Barto。2018年。强化学习：导论。麻省理工学院出版社。[47] Tarik Taleb、Nei Kato 和 Yoshiaki Nemoto。2006年。REFWA：一种适用于低地球轨道卫星网络的高效、公平拥塞控制方案。*IEEE/ACM* 网络汇刊 14, 5 (2006), 1031–1044。[48] Chen Tessler、Yuval Shpigelman、Gal Dalal、Amit Mandelbaum、Doron Haritan Kazakov、Benjamin Fuhrer、Gal Chechik 和 Shie Mannor。2022年。用于数据中心拥塞控制的强化学习。发表于*AAAI*人工智能会议论文集，第36卷。12615–12621。[49] Aashma Uprety 和 Danda B Rawat。2020年。物联网安全的强化学习：全面综述。*IEEE*物联网期刊 8, 11 (2020), 8693–8706。[50] András Varga 和 Rudolf Hornig。2008年。OMNeT++ 仿真环境概述。发表于第一届通信、网络与系统仿真工具与技术国际会议及研讨会论文集。1–10。[51] Qi Wang、Jianmin Liu、Katia Jaffrès-Runser、Yongqing Wang、Chentao He、Cunzhuang Liu 和 Yongjun Xu。2021年。INCdeep：基于深度强化学习的智能网络编码。发表于*IEEE INFOCOM* 会议论文集。1–10。[52] Shangxing Wang、Hanpeng Liu、Pedro Henrique Gomes 和 Bhaskar Krishnamachari。2018年。无线网络动态多信道接入的深度强化学习。*IEEE*认知通信与网络汇刊 4, 2 (2018), 257–265。[53] Keith Winstein 和 Hari Balakrishnan。2013年。TCP ex machina：计算机生成的拥塞控制。*ACM SIGCOMM*计算机通信评论 43, 4 (2013), 123–134。[54] Haitao Wu、Zhenqian Feng、Chuanxiong Guo 和 Yongguang Zhang。2013年。ICTCP：数据中心网络中针对 TCP 的汇聚拥塞控制。*IEEE/ACM* 网络汇刊 21, 2 (2013), 345–358。[55] Chunlei Xu、Weijin Zhuang 和 Hong Zhang。2020年。一种用于软件定义网络路由优化的深度强化学习方法。发表于*CSAE* 会议论文集。[56] Zhiyuan Xu、Jian Tang、Chengxiang Yin、Yanzhi Wang 和 Guoliang Xue。2019年。经验驱动的拥塞控制：多路径 TCP 遇上深度强化学习。*IEEE*通信领域精选期刊 37, 6 (2019), 1325–1336。[57] Zhiyuan Xu、Dejun Yang、Jian Tang、Yinan Tang、Tongtong Yuan、Yanzhi Wang 和 Guoliang Xue。2021年。一种用于经验驱动网络的基于行动者-评论家的迁移学习框架。*IEEE/ACM* 网络汇刊 29, 1 (2021), 360–371。[58] Changhe Yu、Julong Lan、Zehua Guo 和 Yuxiang Hu。2018年。DROM：利用深度强化学习优化软件定义网络中的路由。*IEEE Access* 6 (2018), 64533–64539。[59] Jiuxia Zhao、Minjia Mao、Xi Zhao 和 Jianhua Zou。2021年。一种结合深度强化学习与局部搜索的车辆路径问题求解方法。*IEEE*智能交通系统汇刊 22, 11 (2021), 7208–7218。[60] Lei Zhao、Jiadai Wang、Jiajia Liu 和 Nei Kato。2019年。智慧城市的人群管理路由：深度强化学习视角。*IEEE*通信杂志 57, 4 (2019), 88–93。[61] Chen Zhong、M. Cenk Gursoy 和 Senem Velipasalar。2020年。基于深度强化学习的无线网络边缘缓存。*IEEE*认知通信与网络汇刊 6, 1 (2020), 48–61。[62] Hao Zhu、Yang Cao、Wei Wang、Tao Jiang 和 Shi Jin。2018年。移动边缘缓存的深度强化学习：综述、新进展与开放问题。*IEEE Network* 32, 6 (2018), 50–57。[63] Hao Zhu、Yang Cao、Xiao Wei、Wei Wang、Tao Jiang 和 Shi Jin。2019年。物联网瞬态数据缓存：一种深度强化学习方法。*IEEE*物联网期刊 6, 2 (2019), 2074–2083。