摘 要

网络空间安全形势日趋严峻且面临着一系列的挑战。网络靶场旨在通过虚拟仿真技术创建高度近似于真实网络空间运行机制的网络仿真环境，为网络攻防演练和网络安全新技术验证提供平台支撑，是网络空间安全的重要信息基础设施。网络流量仿真作为网络靶场的重要组成部分，对网络安全新技术验证以及安全策略的评估具有重要的意义。

流量回放技术主要通过解析真实网络环境中的流量数据，并将其无缝、逼真、灵活地注入到网络仿真环境中，解决了传统网络流量仿真中逼真度低的问题，成为了当前网络流量仿真的主流技术。然而，互联网流量具有的协议多样、规模庞大、场景复杂等特点，给流量回放技术带来了挑战。针对上述挑战，本文重点研究了多样化、高性能网络流量回放技术，并构建了大规模网络流量回放仿真系统，提升了网络流量回放模板的多样性、回放方法的多样性以及仿真规模的可扩展性。具体而言，本文的主要研究内容如下：

1）针对互联网流量的协议多样、规模庞大等特点，提出了一种高性能、可扩展的流量模板库构建技术。该技术通过流量数据采集与流量模板生成，可面向多样的真实互联网流量自动生成各类流量模板。针对现有流量数据采集技术在高负载网络环境下采集延迟高、可扩展性不足的问题，提出了基于 XDP 的流量数据采集优化技术；针对流量模板存储中存在的内核存储系统性能不足的问题，提出了基于 SPDK 的流量数据存储优化技术。实验表明，所提出的技术在高负载网络环境下能捕获所有符合条件的流量数据，并进一步生成多样化的流量模板，相较于传统方法，降低了流量采集延迟和模板读取延迟，提升了模板库的并发度。

2）针对流量回放的规模庞大、场景复杂等特点，提出了一种多样化、高性能的网络流量回放技术。设计了基于 DPDK 的高性能流量回放架构，该架构主要通过数据包预缓存、零拷贝和多线程扩展等技术，解决了基于 Linux 内核协议栈回放流量存在的性能瓶颈问题，提升了大规模流量回放的性能。针对无状态型、有状态型、性能测试型等多种场景下的流量回放特点，分别提出了基于时序的无状态流量回放算法、基于状态控制的有状态流量回放算法以及基于 VPP 的矢量流量回放策略，以实现多样化场景下的流量回放，并保证了逼真性与性能。实验表明，所提的方法能够满足多种场景下的流量回放需求与特点，在无状态型方面，提升了回放的时序逼真度；在有状态型方面，能逼真地模拟协议的交互状态；在性能测试型方面，回放吞吐量可以达到虚拟网卡的极限带宽。

3）基于 1）、2）的研究成果，面向大规模网络流量回放需求，设计并实现了大规模网络流量回放仿真系统，可实现仿真实验的全周期管理，提升了易用性。重点探讨了大规模流量仿真场景构建、仿真节点集群控制、仿真实验数据可视化监控等关键技术的设计。基于上述系统，在典型的大规模 DDoS 泛洪攻击、卫星互联网用户行为仿真等方面进行了应用，验证了系统的性能与适用性。

关键词：网络靶场；网络流量仿真；流量回放；多样性；流量模板

Abstract

The network security situation is becoming increasingly severe and facing a series of challenges. The cyber range is intended to create a highly realistic network emulation environment through virtual emulation technology, which is an important information infrastructure for cybersecurity in cyberspace and can provide platform support for network attack and defense exercises and new network security technology verification. Network traffic emulation is an important component of the cyber range and is of great significance for the verification of new network security technology and the evaluation of security strategies.

Traffic replay technology solved the problem of low fidelity in traditional network traffic emulation by analyzing traffic data in real network environments and seamlessly, realistically, and flexibly injecting it into network emulation environments, and became the mainstream technology in current network traffic emulation. However, the diverse protocols, large scale, and complex scenarios of internet traffic pose challenges to traffic replay technology. In response to the above challenges, this article focuses on researching diversified and high-performance network traffic replay technologies, and constructing a large-scale network traffic replay emulation system, which improves the diversity of traffic templates, replay methods, and scalability of emulation scale. Specifically, the main research contents of this paper are as follows:

First, aiming at the challenges brought by diverse protocols and large scale of internet traffic, a high-performance and scalable technology for traffic template library construction is proposed. This technology can automatically generate various traffic templates for diverse real internet traffic through traffic data collection and traffic template generation. An optimization technology for traffic data acquisition based on XDP is proposed to solve the problem of high delay, and insufficient scalability of existing traffic data collection technology in high-load network environments. To address the problem of insufficient storage performance of traditional traffic template libraries, an optimization technology for traffic data storage based on SPDK is proposed. The experiment shows that the proposed technology can capture all eligible traffic data in a high load network environment and further generate diverse traffic templates. Compared to traditional methods, the delay of collecting traffic data and reading traffic template are reduced, the average storage throughput has been improved.

Second, aiming at the challenges brought by large scale and complex scenarios in traffic replay, a high-performance and diversified technology for network traffic replay is proposed. A high-performance traffic replay architecture based on DPDK is proposed, which mainly solves the performance bottleneck problem of traffic replay based on Linux kernel protocol stack through technologies such as packet pre caching, zero copy, and multithreading extension, and improves the performance of large-scale traffic replay. Aiming at the characteristics of traffic replay in various scenarios such as stateless, stateful, and performance testing, a stateless traffic replay algorithms based on timing, a stateful traffic replay algorithms based on state controlled, and a vector traffic replay strategies based on VPP are proposed to achieve traffic replay in diverse scenarios and ensure fidelity and

performance. Experiments show that the proposed method can meet the traffic replay requirements and characteristics in multiple scenarios, and improves the timeliness fidelity of replay compared with traditional methods in the stateless aspect. In the stateful aspect, it can simulate the interaction state of the protocol vividly. In terms of performance testing, the replay throughput can reach the limit of network interface card.

Finally, based on the research results, a large-scale network traffic replay system is designed which focusing on the design of key technologies such as large-scale traffic emulation scenario construction, emulation node cluster control, and emulation experiment data visualization monitoring. Based on the above system, applications have been conducted in typical large-scale DDoS attacks, satellite internet user behavior emulation, and other aspects, verifying the performance and applicability of the system.

Keywords: cyber range; traffic emulation; traffic replay; diversity; traffic template

目 录

第一章 绪论

1.1 研究背景与意义

1.1.1 研究背景

当今社会，随着网络技术的不断发展，网络空间的规模和复杂度不断增加，网络空间安全事件也日益频繁。2021 年初，随着虚拟加密货币的火热，不法分子以垃圾邮件、钓鱼网站等形式大量传播“挖矿”病毒，致使数百万政府、公司、学校和个人的计算机被感染，不知不觉中沦为“矿机”。2021 年 7 月，国家网络安全审查办公室对滴滴公司实施网络安全审查，并于 2022 年 7 月发布审查报告，表明滴滴公司因存在过度收集用户行为信息、违法收集用户手机信息的行为和存在严重影响国家安全的数据处理活动，处以 80.26 亿元的罚款。2021 年底，全球知名开源日志组件 Apache Log4j 被曝存在严重高危漏洞，该漏洞最终能够导致个人信息泄露和远程代码执行。由于 Log4j 框架被各个组织广泛使用，此次漏洞可能是近年来发现的最严重的计算机漏洞之一。2022 年俄乌战争爆发之后，国际黑客组织“匿名者”(Anonymous)官宣正式对俄罗斯政府发动网络战争，先后对俄罗斯关键基础设施进行网络攻击，攻击目标包括交通、能源、政府、军队、银行等，例如工业气体控制系统、俄罗斯联邦储蓄银行、私人银行、国家电视台、军事通讯、政府网站、国防部网站等。层出不穷的网络空间安全事件威胁着网络空间的可靠性及稳定性，习近平总书记 2014 年在重要讲话中指出：“没有网络安全就没有国家安全”，将网络空间安全问题提升到国家战略层面[1]。为了应对网络空间安全在理论研究、技术评估、应用实践等方面的挑战，加强网络安全的建设和管理，网络靶场应运而生。

网络靶场[2-3]是通过仿真技术创建的高度近似于真实网络空间运行机制的重要信息基础设施，可为网络攻防演练和网络安全技术验证提供平台支撑。通过网络靶场可以构建一个可控、可管、可信、可按需定制的网络空间安全环境，从而对网络空间活动和安全能力进行系统研究。依托虚拟化技术[4]实现的云平台[5]可将物理资源进行抽象，从逻辑角度配置资源，提供了优秀的资源整合与管理能力，在规模的可扩展性、场景构建的灵活性等方面具有较强的优势，可作为网络靶场的底层支撑。目前，国内一些高校、研究机构和企业充分借鉴国外先进网络靶场的建设经验，利用各类先进技术，大力推进互联网靶场的研究和建设[6]。

1.1.2 研究意义

网络靶场的研究包括大规模网络仿真、网络流量仿真和用户行为模拟等关键技术。网络流量仿真[7]作为网络靶场的重要组成部分，可将真实网络环境中的流量映射到基于云平台搭建的仿真网络环境中，提供一种按需灵活定制的流量场景复现能力。完善网络流量仿真体系，从理论和实践上加快对网络流量仿真的深入研究，对网络靶场中的网络性能分析评估、产品和技术验证、网络入侵检测、网络攻防演练与研究发展等具有重要的意义。

与基于数学模型[8]合成数据报文的流量仿真技术相比，基于流量回放[9]的流量仿真

技术可将现实网络中采集到的流量数据无缝、逼真、灵活的注入测试网络中，完整、精确地复现流量数据包层面的内容，有效提升网络流量仿真的逼真性。在保证流量数据完整性和回放精确性的前提下，流量回放技术几乎可以一比一复现真实的网络场景，近年来在网络安全、网络测试评估[10]等领域备受关注。

因此，为了完善网络流量仿真体系，提升网络流量仿真的逼真性、灵活性、扩展性，对依托于流量回放的网络流量仿真技术开展进一步研究，具有重要意义和应用价值。

1.2 国内外研究现状

国内外对于网络流量仿真的探索主要围绕仿真模拟平台、流量仿真技术两方面进行。

1.2.1 仿真模拟平台

仿真模拟平台可分为纯硬件的大规模物理实验床、纯软件的仿真模拟软件以及融合软硬件优点的云计算平台。

大规模的物理实验床主要由政府单位或者大型组织构建，PlanetLab[11]是一个由上千物理节点组成的计算机集群，覆盖超过 25 个国家，用于测试各种全球规模的服务，包括文件共享和网络内置存储、内容分发网络、可规模扩展的事件传播、异常检测机制和网络测量工具等。ModelNet[12]是专为真实网络系统进行可重复的大规模实验而设计，用于测试高容量网络、复杂的基础设施软件(存储和虚拟化)以及复杂的网络负载。CERNET[13]是教育部管理的全国性学术计算机互联网络，用于开展下一代互联网研究的网络试验，它以现有的网络设施和技术力量为依托，建立了全国规模的 IPV6 试验床。上述大规模的物理实验床能直接在真实的网络下进行网络攻防演练、测试新型网络技术，但需要耗费巨量的资源，扩展性差，难以推广。

经典的仿真模拟软件主要包括 OPNET[14-15]、QualNET[16]和 NS-3[17-19]。OPNET 和 QualNET 作为商业化的网络仿真软件，可以为网络拓扑以及基础设施间新型通信协议提供设计和分析支持。文献[20]基于 OPNET 仿真环境建立了新型按需组播路由协议的网络模型，通过设置不同的参数和统计量，仿真检验了协议的性能，并证明了所建立协议模型的功能正确性。文献[21]搭建了基于 QualNet 的高低轨混合卫星网络仿真平台，设计了网络拓扑、协议模型体系和分析评估指标体系，对仿真技术解决高低轨混合卫星网络中组网、路由传输等问题的能力进行了验证。但 OPNET 和 QualNET 的价格昂贵，使用成本较高，而且存在版本更新不及时的问题。NS-3 是一个开源的离散事件仿真软件，被广泛运用于教学和学术研究。NS-3 中每个事件都与其执行时间相关联，可以按照时间顺序执行事件来模拟网络活动。文献[22]基于 NS3 搭建了新型的网络架构—确定性网络，重点研究了确定性网络中的显式路由和时延稳定机制。上述仿真模拟软件可以很好地完成小规模的网络流量仿真，但它们只能运行于一台计算机之上，仿真类型单一，无法搭载真实的业务系统与流量，仿真流量数据规模小、逼真性低。

随着虚拟化技术的兴起，云平台应运而生，能很好地折中仿真的资源消耗与逼真

性。虚拟化技术实质是一种资源池化技术，它在底层的物理资源上增加了一层抽象管理层，将有限的物理资源划分为虚拟资源池，并以动态可扩展的方式提供给上层，实现了物理资源的高效利用。以 OpenStack 为代表的云平台，主要依托于操作系统虚拟化和网络虚拟化技术构建。KVM[23-24](kernel-based virtual machine)是目前最成熟的操作系统虚拟化方案，依靠中间层软件 Hypervisor 池化 CPU、内存、磁盘等物理资源，可以为上层隔离出多个完整的操作系统内核。OpenvSwitch(OVS)[25]是一款软件定义网络的虚拟化交换机软件，可用于生产环境，支持跨物理服务器分布式管理、扩展编程、大规模网络自动化和标准化接口。OVS 可以方便管理和配置虚拟机网络，检测多物理主机在动态虚拟环境中的流量情况。基于 OpenStack 云平台，利用 KVM 技术可以创建具有真实操作系统的仿真节点，利用 OVS 技术可以创建接近于真实链路性能的网络拓扑。云平台支持动态扩展、弹性伸缩，可以作为大规模网络流量仿真的有效支撑。文献[26]提出了基于 OpenStack 的 EmuStack 仿真平台，用于仿真大规模、动态、分布式的延迟容忍网络，实现了对链路特性和拓扑的同步、动态、精确仿真。文献[27]提出了基于 OpenStack 的卫星互联网仿真架构，可以实现仿真拓扑的实时重构，并提升了链路动态仿真的同步精度。文献[28]在云平台上回放网络流量，实现了对用户行为流量的仿真，并通过延迟优化技术提升了流量回放的时序逼真度。

1.2.2 流量仿真技术

流量仿真的基本目标是生成可以反映网络基本特征和服务能力的流量，具体分为基于数学模型合成流量和基于真实流量回放两种方法。

基于数学模型合成流量方面，文献[29]改进了分形高斯噪声(FGN)模型的算法，并实现了一个能基于模型生成流量和对生成的流量进行自动检测的系统，该系统能并行且高速地生成自相似的网络流量。文献[30]基于 ON /OFF 模型提出了一个针对 Web 业务的背景流量生成系统，可以生成符合时空特性的背景流量，实现了对背景流量的深度仿真。基于模型驱动生成流量的方法依靠真实网络中流量的数学特征来建立相关的模型，然后通过解析模型合成流量报文，生成流量的逼真性取决于建立的流量模型是否准确。然而目前网络环境复杂多变，难以建立与真实网络场景一致的流量模型，导致基于数学模型生成的流量逼真度有限。

流量回放方法通过采集来自真实网络环境的流量，然后以一定的方式在仿真网络中重放流量，可以完整、精确地复现流量数据包层面的内容，满足流量仿真逼真性的需求。文献[31]设计了一个名为 Monkey 的流量回放系统，可以收集服务器上 TCP 流量，并记录流量的延迟、带宽、响应时间等特征，然后在另一个节点中回放这些流量，以测试服务器的性能。文献[32]设计了一个用于流量回放的开源工具 D-ITG，它可以准确描述流量数据包级别的时间和规模特性，并支持回放 IPv6 流量。文献[33]设计了名为 OctReplay 的流量回放系统，该系统在硬件支持下提高了流量回放的性能。文献[34]提出了一种有状态 TCP 流量回放方法，通过模拟 TCP 协议栈来维持回放的状态，并基于收发平衡来减少状态判定开销，以提高回放性能。但上述方法只适用于单台物理机回环或者串联一台 DUT 设备的场景。文献[35]研究了面向天地一体化网络的高并发用户

行为仿真技术，提出了基于误差补偿的流量回放方法，但该研究更侧重恶意流量仿真。文献[36]通过计算物理网络与虚拟网络的相似性实现了最佳的 IP 映射，然后在云平台上的缩小网络上实现了多机互动的流量回放，但该研究面向低速率流量场景，不适合仿真目前的网络环境。文献[37]提出了一种基于数据包时序的流量回放方法，并通过流量去重、过滤和映射等数据操作，提升了流量回放的准确性，但该研究的流量回放方式较为单一且回放速度不足。

1.2.3 面临的问题

通过上述研究可以得知，相较于纯硬件或者纯软件的方式，云平台具有弹性伸缩的优势，已成为主流的网络流量仿真平台。相较于数学模型合成流量的方法，流量回放技术能复现数据包层面的内容，更具逼真性，依托于流量回放在云平台上进行网络流量仿真已成为一种趋势。但是当前互联网环境复杂多变，现有的流量回放方案仍然存在一些挑战，具体体现在：

1）流量规模庞大。流量回放以预先捕获的流量数据作为流量模板，从而保证后续流量生成的逼真性。目前网络技术高度发展，流量数据的种类、数量也随之剧烈增长，亟需建立流量模板库进行统一的管理。流量模板库的构建过程中，首先需要解决流量模板的获取问题，以提升协议模板的多样性，具体为如何可靠、高效地获取互联网中协议流量，并进一步生成多样化的流量模板；其次需要考虑优化流量模板库的存储性能，以提升仿真规模的可扩展性，具体为如何将流量模板高效的提供给仿真节点。

2）流量场景复杂。目前基于流量回放的流量仿真研究，大多面向低速网络场景仿真，在仿真高速流量时逼真度不够，且仿真节点的回放方式较为单一，缺乏扩展性。随着网络环境逐渐复杂化，流量回放节点应支持回放多样化、高速率的网络流量，支持复杂网络场景的灵活、可扩展加载，在低速和高速流量仿真时均能准确控制回放的执行过程，逼真地还原现实网络环境。

1.3 本文研究内容

根据上述分析，针对当前网络流量回放方案在多样性、可扩展性、逼真性等方面存在的不足，以完善网络流量仿真体系为研究目标，深入研究了多样化、高性能网络流量回放技术。具体研究内容包括：

首先，针对互联网流量的协议多样、规模庞大等特点，提出了一种高性能、可扩展的流量模板库构建技术。首先，为了提升流量模板库构建的性能与多样性，针对现有流量数据采集技术在高负载网络环境下采集延迟高、可扩展性不足的问题，提出了基于 XDP 的流量数据采集优化技术，旨在高负载网络环境下捕获所有符合条件的流量数据，并进一步生成多样化的流量模板；其次，为了提升流量模板库存储的性能，提出了基于 SPDK 的流量数据存储优化技术，旨在解决内核存储系统性能不足的问题。最后，通过实验验证该技术的可行性和性能。

其次，针对流量回放的规模庞大、场景复杂等特点，提出了一种多样化、高性能的网络流量回放技术。首先，针对采用 Linux 网络协议栈回放流量性能不足的问题，

提出了一个基于 DPDK 的流量回放架构，通过数据包预缓存、零拷贝和多线程扩展等技术提升了大规模流量回放的性能。基于此架构，面向无状态型流量回放场景，提出了基于时序的无状态流量回放算法，并设计了多种回放速率控制机制；面向有状态型流量回放场景，提出了基于状态控制的有状态流量回放算法；面向性能测试型流量回放场景，提出了基于 VPP 的矢量流量回放策略。最后，通过实验验证该技术的逼真性与性能优势。

最后，设计并实现了一个大规模网络流量回放系统，着重介绍了系统架构、仿真流程和系统关键技术的设计。基于该系统，实验人员可以根据不同的仿真任务需求构建自己的仿真场景，实现仿真实验的全周期管理。

1.4 本文结构安排

本文余下各章的组织结构关系如下图 1-1 所示：

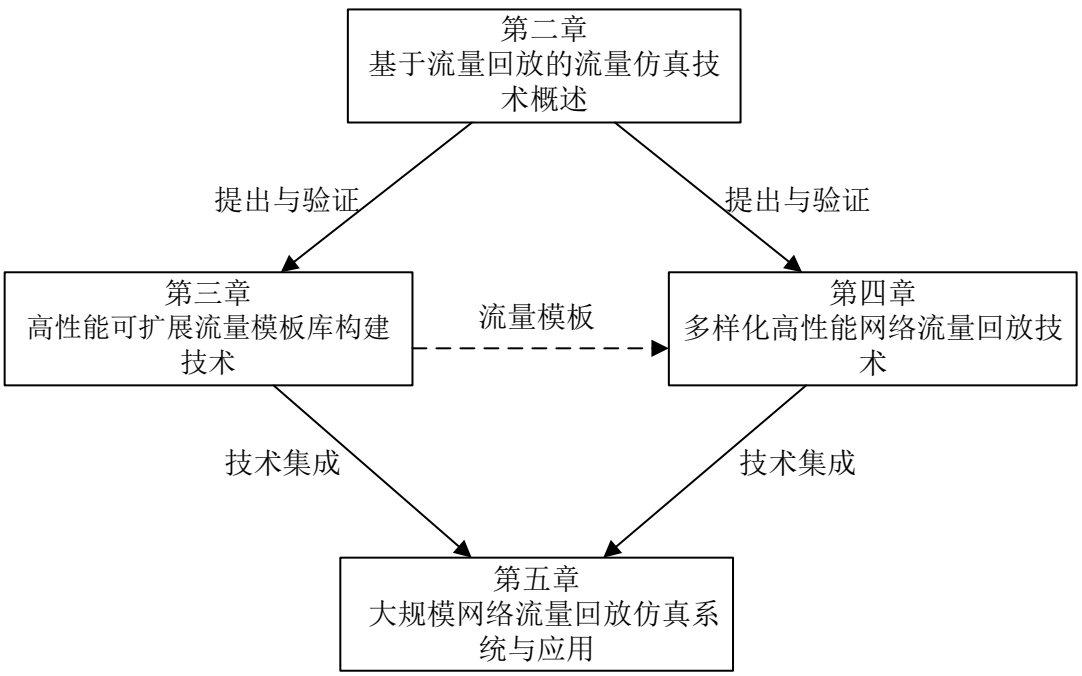

图 1-1 本文章节安排

第二章，基于流量回放的流量仿真技术概述。首先介绍了网络流量仿真的特点和目标，优选契合仿真需求的流量回放技术作为底层技术支撑，其次，介绍基于流量回放的流量仿真所涉及的相关技术，包括流量数据采集与存储技术和流量报文再生成技术；再次，介绍为网络流量仿真提供平台支撑的虚拟化与云计算技术；最后，结合现有技术现状，分析当前流量回放方案存在的问题。

第三章，高性能可扩展流量模板库构建技术。首先阐述构建流量模板库的必要性，对构建流量模板库需要解决的问题进行分析，包括提升多样化流量模板获取的高效性、扩展性以及流量模板库的存储性能。然后，为了解决多样化流量模板获取问题，针对现有流量数据采集技术在高负载网络环境下采集延迟高、扩展性不足的问题，提出了基于 XDP 的流量数据采集优化技术；为了提升模板库的存储性能，为仿真节点提供高效的存储服务，提出了基于 SPDK 的流量数据存储优化技术。最后，对该技术的功能

与性能进行了实验验证。

第四章，多样化高性能网络流量回放技术。首先论述了当前流量回放技术在灵活性、可扩展性和性能方面的不足，然后针对上述问题提出了一个基于 DPDK 的高性能流量回放架构，以减少内存拷贝、上下文切换和锁竞争带来的开销。基于此架构，面向无状态和有状态两种流量回放场景，提出了基于时序的无状态流量回放算法和基于状态控制的有状态流量回放算法。进一步，针对高速流量生成的需求，提出了基于 VPP 的矢量回放策略。最后，设计了回放实验，对所提出技术的灵活性、可控性、逼真性、性能进行验证。

第五章，大规模网络流量回放仿真系统与应用。基于第三章和第四章的相关技术，针对大规模仿真场景部署需求和仿真节点集群控制需求，设计了相应的模块，以实现大规模流量仿真场景的全周期管理，并通过可视化图表的形式，直观地展示仿真任务执行数据。最后，在典型的大规模 DDoS 攻击、卫星互联网用户行为仿真等方面进行了应用，验证了系统的性能与适用性。

第六章，主要结论与展望。对本文所研究和完成的工作进行总结，并对下一步工作进行展望。

第二章 基于流量回放的流量仿真技术概述

2.1 引言

本章节对所涉及的理论与技术展开研究。首先，阐述了本文所研究网络流量仿真的目标和相关技术，选取流量回放作为仿真方法的优势和必要性；其次，对流量回放涉及的相关技术进行介绍，包括流量数据采集与流量数据存储技术、流量报文再生成技术以及虚拟化与云计算技术；最后，结合现有技术现状，分析当前流量回放方案仍面临的问题，从而为后续研究与实践提供理论基础。

2.2 网络流量仿真技术概述

随着互联网的发展，网络空间逐渐成为一个复杂的实体，充斥着各种应用、用户行为交互产生的流量。网络流量仿真可将真实网络环境中的流量映射到网络仿真环境中，为网络规划、建设提供可靠、高效、客观的理论和定量依据，对新兴网络与安全技术的评估和应用具有重要的意义。网络流量仿真的基本目标是生成可以反映网络基本特征和服务能力的流量，具体仿真方法包括从宏观角度建立数学模型的模型驱动方法和从微观角度复现数据包层面特征的流量回放方法。

早期网络流量具有自相似特性，涌现出了诸如重尾分布的 ON/OFF 模型[38]、FBM/FGN 模型[39]和 FARIMA 模型[40]等用于构造自相似数据序列的模型，这些模型生成的流量具有宏观上的长相关性，但无法反映流量的微观特征。基于模型驱动的方法可以刻画某一段时间内网络流量的特征，但当前网络环境复杂多变，需要根据网络状态，定期对模型进行修改，导致生成的流量逼真度有限，难以全面地反映网络流量的特征，存在一定的局限性。流量回放方法生成的流量直接来源于真实网络环境，通过报文重构再生成的方式，可以完整、精确地复现流量数据包层面的内容。在保证流量数据完整性和回放精确性的前提下，流量回放几乎可以一比一复现真实的网络场景，更符合当前网络流量仿真的高逼真需求。后续流量回放从宏观上讲，指基于流量回放的流量仿真，从微观上讲，指回放流量报文这一行为。

流量回放以预先捕获的流量数据作为流量模板，为后续流量再生成提供逼真性保障，目前网络技术高度发展，流量数据的种类、数量也随之剧烈增长，亟需建立流量模板库进行统一地管理。但在流量模板库构建过程中仍然存在多样化流量模板获取困难和流量模板库存储系统性能不足的问题，这严重阻碍了流量回放技术在各类复杂仿真场景下的应用以及流量回放规模的扩大。针对互联网流量的协议多样、规模庞大等特点，构建流量模板库需要实现高效、可扩展的获取流量模板，以提升协议模板的多样性；需要优化流量模板库的存储性能，以提升仿真规模的可扩展性；前者依托于流量数据采集技术，后者依托于流量数据存储技术。另一方面，流量回放通过解析流量模板再生成的方式，以复现现实的网络场景。随着网络环境逐渐复杂化，需要研究提升报文再生成过程可控性、可扩展性的技术，以最大化发挥流量模板的价值，提升流量回放的多样性、逼真性。此外，相比 NS3 等仿真模拟软件和实物仿真平台，基于虚

拟化技术实现的云平台，能很好地折中仿真的资源消耗与逼真性，为流量回放提供平台支撑，需要对云平台涉及的相关技术进行研究。

因此，本章后续部分首先对流量数据采集与流量数据存储技术、流量报文再生成技术涉及的相关技术进行剖析，然后介绍虚拟化和云计算技术，最后分析当前流量回放方案存在的问题。

2.3 流量数据采集与流量数据存储技术概述

构建流量模板库需要实现高效、可扩展地获取流量模板，提升协议模板的多样性；需要优化流量模板库的存储性能，提升仿真规模的可扩展性。当前网络环境流量协议多样、数据规模庞大，高性能的流量数据采集技术可以提升流量模板的覆盖率和多样性；高性能的流量数据存储技术可以提升流量模板库并发度、降低响应延迟，提升仿真规模的可扩展性，因此本节介绍流量数据采集和流量数据存储涉及的相关技术。

2.3.1 流量数据采集技术

在介绍流量数据采集技术之前，需要对数据报文的收发过程有一个清晰的认识。目前，Linux 操作系统被广泛应用于学术界和工业界的研究，本文的所有工作也均在 Linux 操作系统下进行，Linux 操作系统内核接收数据报文的过程如图 2-1 所示，网卡硬件在收到数据包后，首先通过 DMA 的方式把数据包拷贝到驱动层的缓冲区，然后向上传递给协议层，协议层进行一定的处理后，将数据包送至应用程序。发送过程与之类似，接下来的相关技术也均以接收角度来进行介绍。

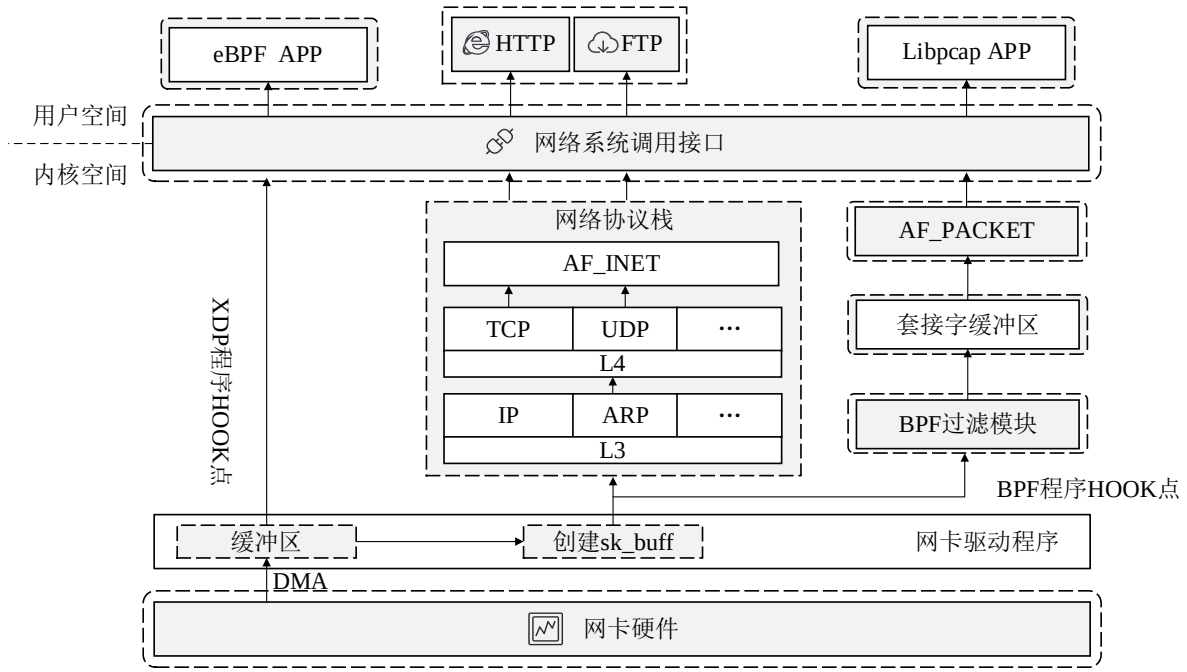

图 2-1 数据包接收架构

1）Libpcap 技术

Libpacp[41]是一款著名的网络数据包捕获库，Tcpdump[42]和 Snort[43]等众多数据包捕获和分析软件都是以 Libpcap 为基础进行开发。如上图 2-1 所示，AF_PACKET 是 Linux 网络协议栈的原始套接字，被 Libpcap 用作底层捕获模块。AF_PACKET 初始化时会在

IP 层的底部 Hook 一个钩子函数，函数运行时将 AF_INET 协议族的所有数据包转发给 BPF(berkeley packet filter)[44]过滤模块，对捕获的数据包按照既定的规则进行过滤，过滤后的数据包存储在 AF_PACKET 的套接字缓冲区，最后被传递给应用层的 Libpcap 程序。

2）eBPF 技术

BPF 已在 1）中有所提及，全名为伯克利数据过滤器。早期的数据包过滤技术是将数据包复制到用户空间再进行过滤，而 BPF 的思想是直接将过滤程序之间插入内核，以减少无用的数据包拷贝。BPF 包含一套由伪机器码组成的字节码，用于编写 BPF 程序，BPF 程序运行时会通过 JIT(just in time)将字节码编译为机器能够执行的机器码。由于先进的设计思想，近年来 BPF 的功能、指令集得到不断的扩充，形成了一套新的框架—eBPF[45](extend BPF)。eBPF 的应用场景不再局限于网络，在安全领域、内核追踪与观测也有着出色的表现，eBPF 的架构如图 2-2 所示：

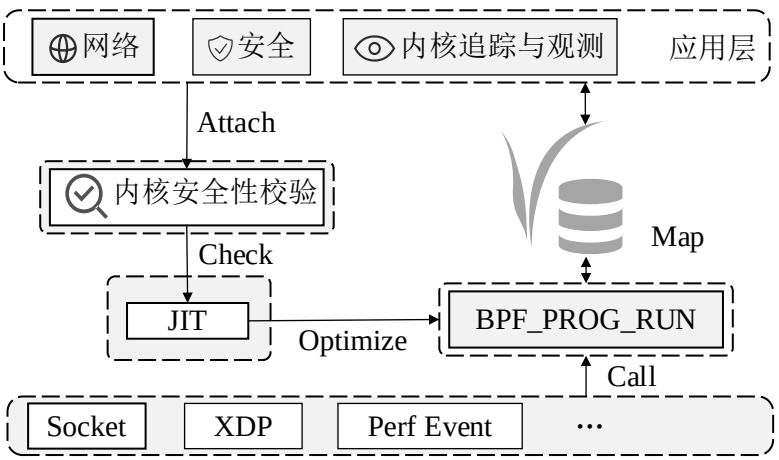

图 2-2 eBPF 架构

eBPF 的扩展主要集中在以下部分：复杂指令集，eBPF 实现了一套新的指令集，包括 87 条 64 位指令和 10 个 64 位寄存器，以支持构建更多 eBPF 程序；安全校验机制，eBPF 程序在运行之前，会经过内核中的校验器审核，以防止恶意代码的注入；Map 机制，Map 是运行在内核 key/value 型数据库，Map 的数据由用户空间和内核共享，可以方便的、高效的在用户程序和内核之间传递信息；方便的程序编写方式，相比于 BPF 程序，eBPF 程序支持由高级语言 C 语言编写，然后由 LLVM 等编译器后端编译为 eBPF 字节码，近年来诸如 BCC、XDP 等集成工具库的出现更是进一步简化了 eBPF 程序的编写和执行。

BCC[46](BPF Compiler Collection)，是一系列内核跟踪和操作程序工具包的集合，主要组件包括一个基于 LLVM 的 C 语言包装器和 Python 前端。BCC 使 eBPF 程序更易于编写，适用于性能分析和网络流量控制等诸多任务。在 BCC 以前，开发人员使用 C 语言编写 eBPF 程序，然后手动使用 LLVM 编译这个程序，将其注入内核。以 C 语言编写 eBPF 程序的运行流程十分复杂，有时需要直接在 Linux 内核源代码树中编译。为了解决这个问题，BCC 提供了一系列的编译工具链，能自动化地编译、执行 eBPF 程序，极大地提升了开发人员的工作效率。BCC 提供的 Python 前端可以将以 C 语言编写

的 eBPF 程序作为字符串嵌入 Python 程序中，并从编译器接收程序有效性的反馈。目前，BCC 工具库已经集成了以下组件，1）端对端的 BPF 工作流，用于与其他模块灵活集成；2）Python 前端，通过简单的 API 编写、运行复杂的 eBPF 程序；3）套接字过滤器、tc 分类器和 kprobes 的示例；4）一系列用于跟踪正在运行系统的独立工具。

XDP[47](eXpress Data Path)是一款在操作系统内核实现快速、可编程包处理的框架，基于 eBPF 机制实现，采用 C 语言编写。XDP 系统基于 Linux 内核运行，可以很方便地与现有系统集成，复用已有的网络基础设施，包括路由表、高层协议栈等。XDP 程序运行时具有充分的隔离性，不会侵犯其他应用的资源空间，保持了内核的安全边界。与 Libpcap 类似，XDP 程序的启动也依靠钩子函数，目前 XDP 钩子函数的 Hook 点分布在网卡硬件(Device Hook)、网卡驱动(Driver Hook)、内核协议栈(Generic Hook)，程序性能也逐步降低。Device Hook 模式的 XDP 程序，处理过程直接 offload 到网卡，程序运行过程不需要 CPU 的参与，具有最高的效率，但是成本昂贵，目前只有有限的网卡硬件支持 XDP。Generic Hook 的钩子函数位于内核协议栈中，不需要专门的网卡或驱动支持，任何支持 eBPF 的 Linux 内核都可以在此模式下运行 XDP 程序，此模式下的 XDP 程序具有最强的通用性，但性能也最差，一般应用于学习场景。目前，在生产环境中应用最为广泛的是 Driver Hook 模式，如上图 2-1 所示，此模式的钩子函数位于设备驱动中，能在数据帧分配 sk_buff 结构体之前对数据进行处理，此时未进行任何解析数据包的操作，具有最高效的软件性能。

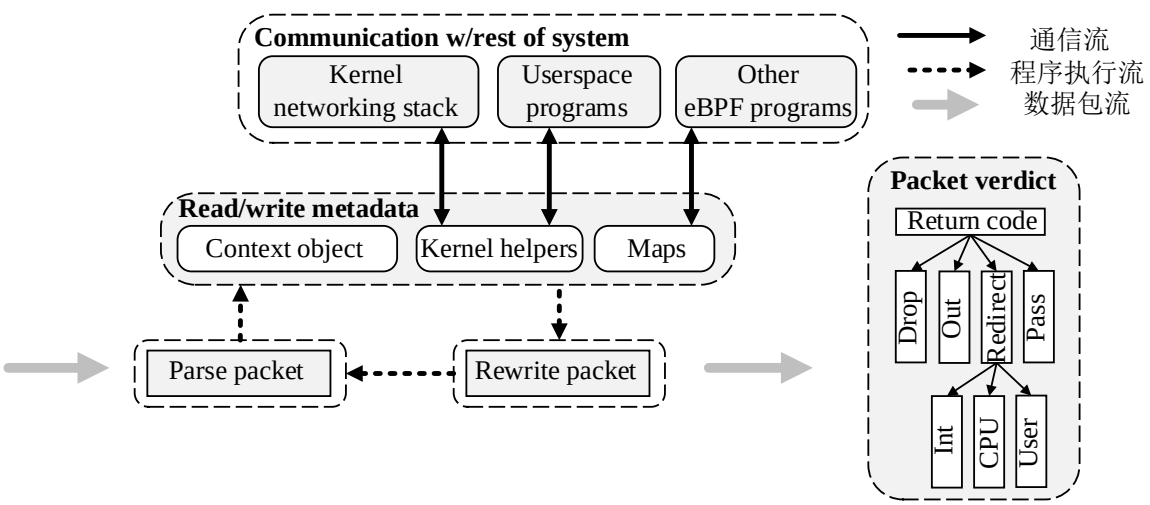

图 2-3 XDP 程序执行流

Driver Hook 模式下的 XDP 程序执行流如上图 2-3 所示。第一步，网卡驱动收到一个数据包后，触发 XDP 的钩子函数，钩子函数会提取包头中的信息(例如 IP、MAC、Port、Protocol 等)，传递给 XDP 程序一个上下文对象，其中包括指向原始数据包的指针和描述这个数据包的元数据。第二步，XDP 程序收到数据包后，根据数据包的信息对一些资源的元数据进行读取或更新，例如访问数据包所对应的内核路由表，更新 key/value 型数据库 Maps 中的 value。第三步，接下来如果有需要，XDP 程序可以对数据包的任何部分进行更改，包括添加和删除包头。此时 XDP 程序可以执行封装和解封装操作，重写数据包的 IP 地址字段，然后根据内核路由表将数据包转发出去。修改后

的数据包需要借助内核的函数重新计算校验和，实现对内核基础设施的复用。最后一步，对这个数据包进行最后的判决，确定数据包接下来的操作。判决结果包括丢弃这个数据包 XDP_DROP、通过接收的网卡重新发送出去 XDP_TX、允许进入内核协议栈 XDP_PASS 和重定向 XDP_REDIRECT。XDP_REDIRECT 支持将数据包转发至另外的网卡发送出去、给指定 CPU 进一步处理和转发给用户态的 Socket 进一步处理。XDP 程序执行流的前三步(读取数据包、处理元数据、重写数据包)可以是任意顺序，并且支持多层嵌套。不同的 XDP 程序之间还可以通过尾调用移交控制权，以达到数个小的 XDP 程序组合成大型 XDP 程序的功能，方便模块之间的解耦。目前，BCC 已经集成了 XDP 模块，在 BCC 中可以直接编写 XDP 程序，将其安全的注入内核，方便快捷地实现高速、可编程的数据包处理。

2.3.2 流量数据存储技术

传统的数据存储方式如图 2-4 所示，从下至上可分为硬件层、内核文件系统、虚拟文件系统、应用层。用户在读取文件数据时首先需要陷入内核，触发系统调用，然后经过虚拟文件系统、文件系统的调度，最终从本地磁盘设备或网络磁盘设备获取所需资源，整个过程十分漫长。由于成本和技术的限制，早期数据存储以机械硬盘为主，相比于访问机械硬盘的开销，内核软件开销在整个流程的占比非常小。近年来随着 NVMe 固态硬盘的兴起，存储硬件的并发度和延迟得到了极大地改进，基于内核的读写方式不能发挥 NVMe 固态硬盘的性能优势，访问开销的 25%消耗在内核软件栈上[48]。

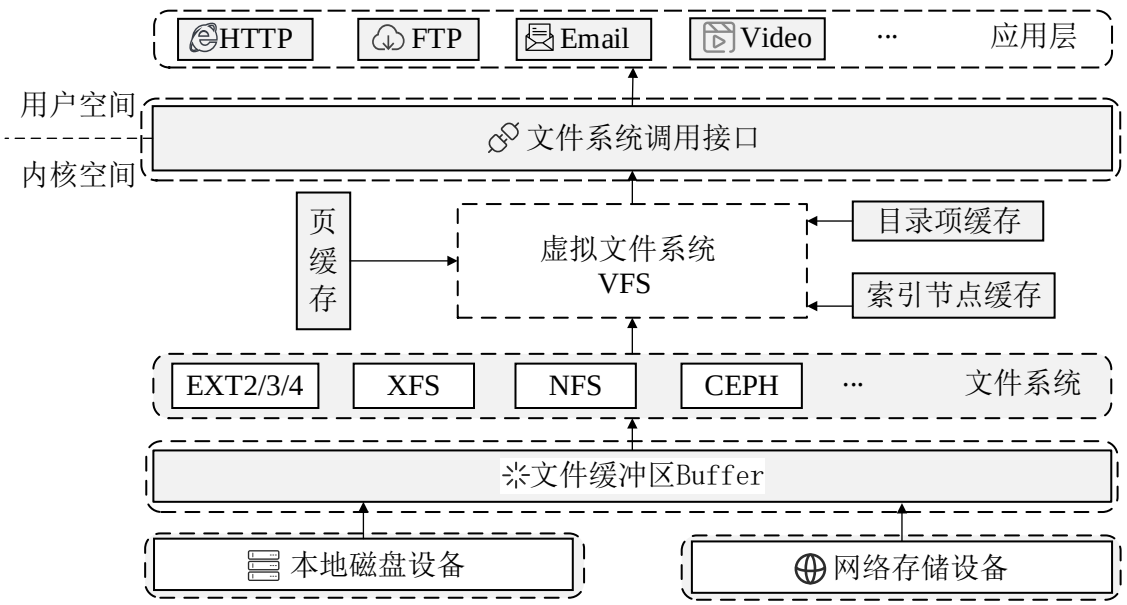

图 2-4 内核存储架构

1）SPDK 技术

为了提升 NVMe 固态硬盘软件栈的性能，构建高性能的存储服务，Intel 提出了存储性能开发套件 SPDK[49](storage performan ce development kit)，将整个存储过程由内核空间迁移到用户空间。SPDK 是一套包含一系列组件的工具库，使用者可以在 SPDK 的基础上编写高性能、可伸缩、用户模式的应用程序。SPDK 具体架构如图 2-5 所示：

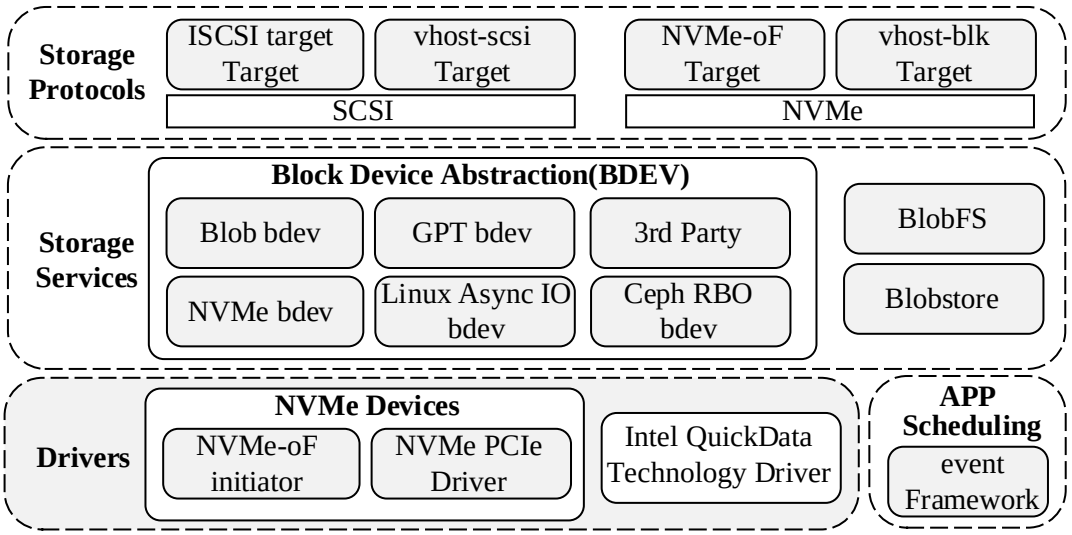

图 2-5 SPDK 架构

SPDK 的存储服务分为三层，Bdev、Blobstore 和 BlobFS。Bdev 是对底层用户态驱动封装而成的块设备库，块设备是一种支持固定大小数据块读写的存储设备，可以对应逻辑上或者物理上的设备，块(block)的大小通常为 512 字节或 4096 字节。Blobstore 用于实现 blob 对象的分配与管理，blob 由多个 cluster 连接而成的链表构成，cluster 由多个连续的 page 构成，cluster 的大小通常为 1MB，page 是 Blobstore 中最基础的存储单元，由多个连续的 block 组成，page 的大小通常为 4KB。BlobFS 是在 Blobstore 的基础上封装的轻量级文件系统，其中的文件与 Blob 对应，提供常用文件接口，如 read、write、open、delete 等，用于支持更上层的应用。

SPDK 提供了一个基于事件驱动的编程框架。在这套框架中设计了如图 2-6 所示的线程模型，各线程间互相独立，以传递消息的方式进行线程同步，消除了共享引起资源竞争，具有良好的扩展性。这套框架提供了三个主要概念，反应堆 reactor、事件 event 和轮询者 poller。其中，reactor 是核心，用于处理传入的事件；event 表示一个任务，可以在线程间传递消息，用于跨线程通信；poller 与 event 类似，需要在 reactor 上注册，但会周期性的执行。

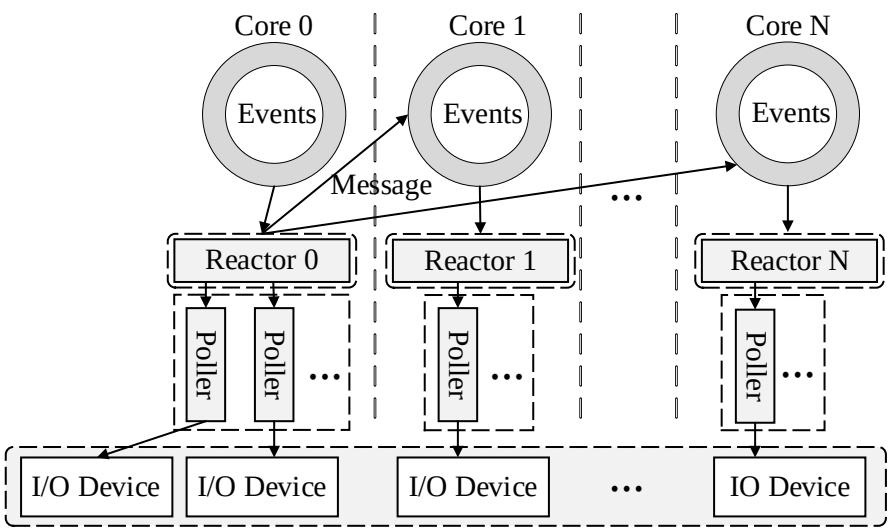

图 2-6 SPDK 线程模型

2）RPC 技术

RPC[50](remote procedure call)技术全称为远程过程调用，是一种在本地机器调用远端函数和过程的通信技术，常用于分布式系统中。RPC 的整个生命过程如图 2-7 所示，涉及到用户代码、序列化、数据组织、压缩、协议等。uRPC、sRPC 等 RPC 框架通过对其中序列化、压缩、协议等技术细节的封装，让使用者更专注于用户代码的编写，以简化分布式系统的通信过程。根据 RPC 的生命周期，RPC 框架可划分为数据处理模块、服务发布模块、服务调用模块。

数据处理模块用于实现数据的传输与交换，包括网络传输、对象的序列化与反序列化、数据封装。网络传输是 RPC 框架最基础的功能，服务提供方和服务调用方需要通过网络传输完成信息交互。序列化是指将对象转换成二进制数据，因为网络传输的数据为二进制格式，反序列化即将二进制数据还原为对象。数据封装是指 RPC 通信双方根据拟定好的数据格式来封装请求数据与响应数据为对象。

服务发布模块和服务调用模块是 RPC 框架的核心模块，服务发布方在服务发布模块上注册服务，将服务与函数或者过程映射后发布出去，并通知服务调用方；服务调用方通过网络发送服务调用请求，经过服务调用路由后选出最合适的服务提供方，解析请求并作出响应。

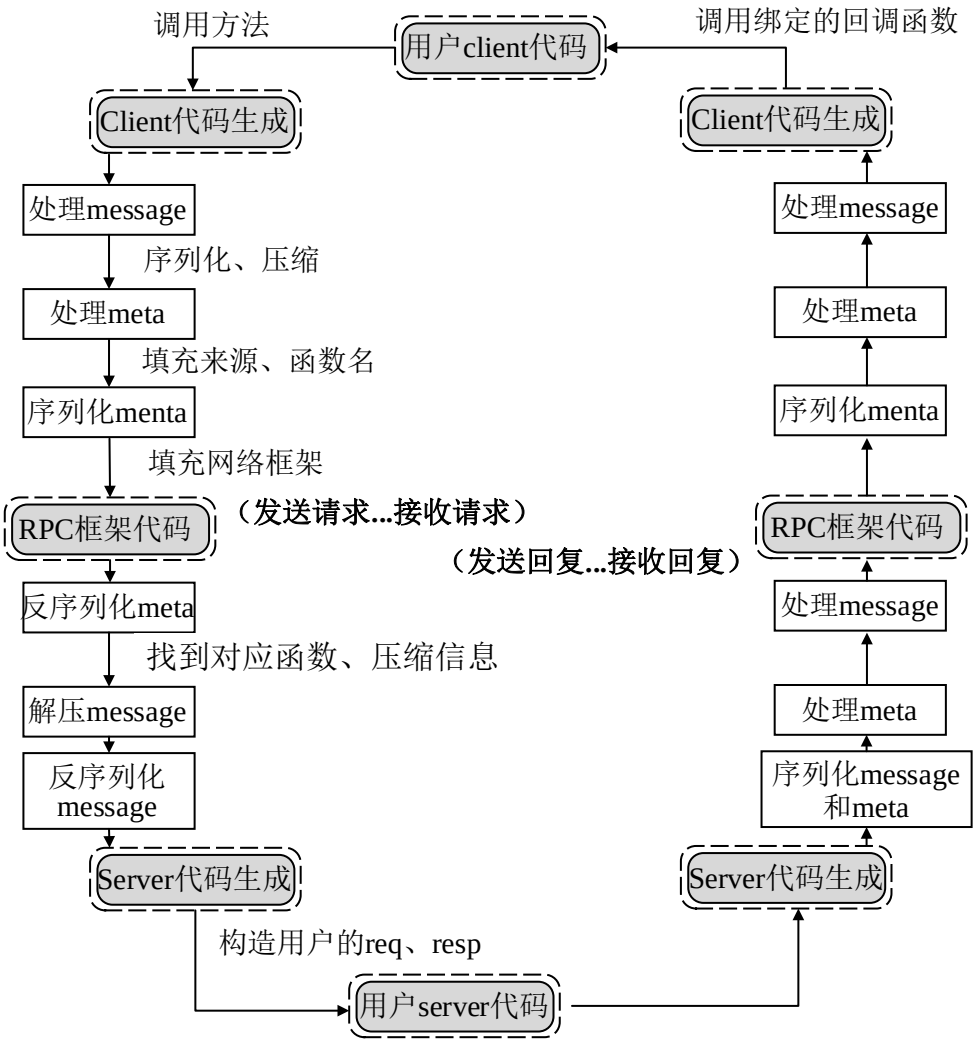

图 2-7 RPC 生命周期

2.4 流量报文再生成技术概述

流量回放的最终步骤是解析流量模板，再生成能反映现实网络场景的流量报文。流量报文再生成包括流量报文重构技术和流量报文回放技术。多样化的流量报文再生成方法可以拔高流量模板的价值，提升流量回放的逼真性和多样性，因此本节对流量报文重构和流量报文回放涉及的相关技术进行介绍。

2.4.1 流量报文重构技术

为了使流量数据在复杂的仿真网络中流转，必须重写流量报文的二三层字段，对 IP、MAC 地址进行调整，进行 DDoS 攻击、有状态连接测试时，还需要对数据包进行更细粒度地修改。常用的报文重构库有 Libnet[51]和 Scapy[52]等。

Libnet 是一款由 C 语言编写的小型接口函数库，封装了操作系统底层网络编程细节，可以方便地构造低层网络数据包。但 Libnet 不支持编辑 IPv6 格式的数据包，仅支持 15 种 IPv4 协议数据包的重构，应用范围较为狭窄。

Scapy 是一款由 Python 语言编写的交互式数据包处理库，使用门槛低且功能强大，能轻松解析和构造海量协议的数据包。如图 2-8 所示，Scapy 采用分层的设计思想，通过逐层堆叠构造完整的数据包，可以实现对数据包参数细粒度地修改，例如 IP 层的 TTL、片偏移，TCP 层的序列号、端口号、窗口大小等。Scapy 在支持常见协议的同时，还具备非常强的扩展性，用户可以通过添加协议图层来为 Scapy 增加新的协议支持。

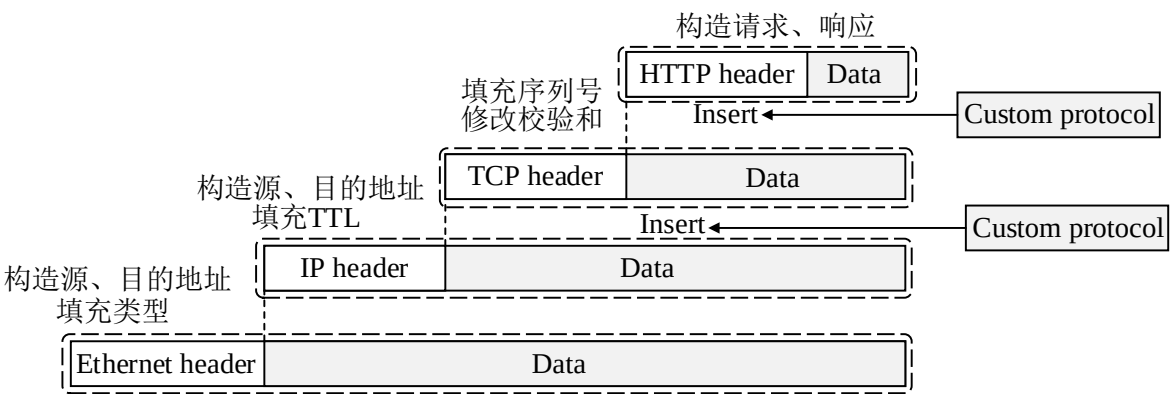

图 2-8 Scapy 设计思想

2.4.2 流量报文回放技术

目前，流量报文回放方式可分为基于操作系统内核的方式和内核旁路的方式。

1）基于内核的流量报文回放工具

操作系统内核收发报文的过程在 2.4.1 节已进行详细的介绍，目前基于内核的流量报文回放工具有 Tcpreplay[53]、TCPCopy[54]、GoReplay[55]等。Tcpreplay 是一款经典的开源流量回放工具，用于回放 Tcpdump 和 Wireshark 捕获的 pcap 格式的流量文件。Tcpreplay 支持编辑数据包的 2-4 层内容，广泛应用于交换机、路由器、防火墙等网络设备的压力测试。TCPCopy 是网易开发的基于请求复制的 TCP 流量回放工具，用于测试真实服务器的应用程序。TCPCopy 由 tcpcopy 和 intercept 组成，tcpcopy 可以捕获在线的真实流量并将其发送给目标服务器，intercept 部署在辅助服务器上，可以将服务器的响应信息转发给 tcpcopy。GoReplay 是一款用 GO 语言开发的网络监控工具，可以捕

获真实的流量数据，支持对流量跟踪、监控和详细分析，逼真地复现请求多样性、网络延迟和资源占用。但这些基于 Linux 内核实现的软件，面向的是通用的传输场景，在回放报文的性能上略显不足，且回放功能固化，缺乏扩展性。

2）DPDK 技术

以 DPDK[56]为代表的内核旁路技术，通过自定义的驱动独占网卡的使用权，在数据包收发和处理阶段绕过内核网络子模块的参与，直接将数据包传递到用户空间。DPDK 是 intel 提出的数据平面开发套件，由一系列核心组件构成，如图 2-9 所示。环境抽象层 EAL 是 DPDK 运行的基础，负责管理硬件和内存空间等低级资源，并提供了一系列隐藏了环境细节的通用接口。EAL 层初始化时首先会执行 PCI 探测程序，解除目标网卡与内核驱动的绑定，进而绑定 DPDK 驱动程序，实现对网卡硬件的独占；其次，使用内核提供的 mmap 函数在巨页上申请一片空间，提供给 DPDK 服务层使用(如 libret_mempool)；最后，根据用户参数设置网卡端口信息，为端口配置收发包队列的数量、大小，并为每个队列分配内存。内存池 librte_mempool 由一组大小相同的对象缓冲区组成，由名称唯一标识，使用 mempool 操作来存储空闲对象。对象缓冲区 librte_mubuf 用于存储单个对象，对象通常是网络数据包，但也可以是控制数据、事件等任何数据。环形缓冲区 librte_ring，用于管理已分配的对象缓冲区和在线程之间传递消息。定时管理器 librte_timer 为 DPDK 的执行单元提供定时器服务，以实现回调函数的异步执行。

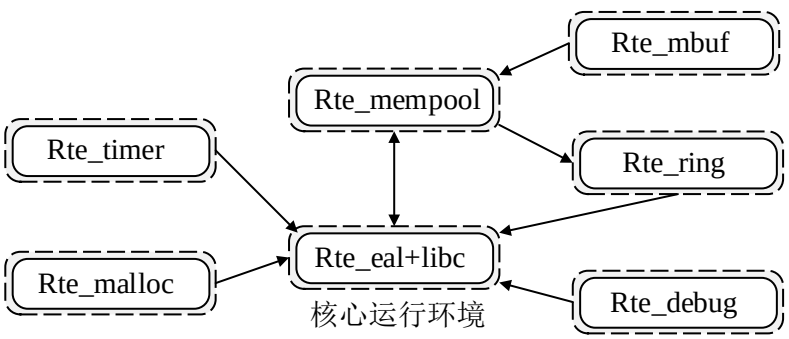

图 2-9 DPDK 库组件

2.5 虚拟化与云计算技术概述

虚拟化与云计算技术可以在平衡资源消耗的前提下，逼真地复现仿真节点和仿真网络。依托于虚拟化和云计算技术进行流量仿真具有灵活可扩展、弹性伸缩的优势。

2.5.1 虚拟化技术

虚拟化是将底层的物理资源抽象分割为虚拟逻辑资源和统一管理虚拟资源的技术。在非虚拟化的系统中，同一时间只能运行一套操作系统，这套操作系统独享所有的硬件资源，常常存在物理资源利用率不足的问题。虚拟化技术可以将抽象后的虚拟资源隔离，按需分配给不同的实体，以达到在一套硬件上运行多个操作系统的目的，能最大化地利用物理资源。在虚拟化的系统中，通常包含一个虚拟机监控器(virtual machine monitor，VMM)和多个虚拟机(virtual machine monitor，VM)，VMM 又称为 Hypervisor，

向下统一管理、隔离实际的物理资源，向上为不同的 VM 提供逻辑资源；VM 拥有完整的操作系统，相互之间无法感知，独享分配的硬件资源。

按照虚拟化技术实现的方式可分为软件虚拟化和硬件虚拟化。软件虚拟化即通过诸如 QEMU[57]等软件模拟实现 VMM，QEMU 能将虚拟机支持的指令翻译成底层物理机的对应指令，以达到在虚拟机中运行任意操作系统的功能。软件虚拟化具有最强的兼容性，但纯软件翻译指令性能不足。硬件虚拟化又称硬件辅助虚拟化，依靠底层物理 CPU 辅助实现(如 IntelVT 技术)，该技术能对 CPU、I/O 设备、网络的虚拟化进行硬件加速，提升性能。

目前最为成熟的虚拟化技术是 KVM，KVM 实际上是内核的一个模块，依靠硬件辅助实现，运行后可以将内核转化为 Hypervisor。KVM 支持对 CPU、磁盘、显卡、网卡等硬件设备的虚拟化，但只能模拟部分 I/O 功能，因此常与 QEMU 同时使用。KVM/QEMU 架构如图 2-10 所示，当虚拟机执行操作时，首先将控制权转交给 KVM，KVM 进行判断，若为 I/O 操作则将控制权转交给 QEMU，否则直接执行指令。

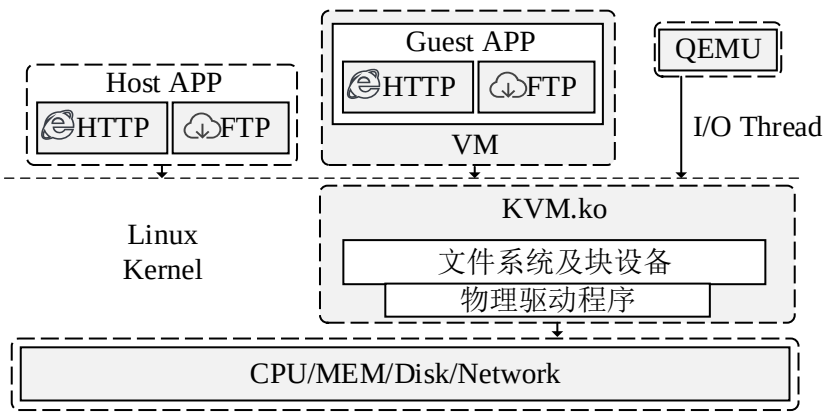

图 2-10 KVM/QEMU 架构

2.5.2 云计算技术

云计算是一种新型的服务提供模式，旨在对互联网上庞大的计算资源、存储资源、数据资源和应用资源进行统一的管理和调度，根据用户的需求动态分配资源，实质是一个具有高灵活性、高可靠性、高扩展性、高资源利用率的分布式系统。云计算提供的服务可分为三类，基础设施即服务(IaaS)、平台即服务(PaaS)、软件即服务(SasS)。这三种服务类型构建成一个服务堆栈，IaaS 位于最底层，中间层为 PaaS，SasS 位于最顶层，抽象程度依次提升。

云平台是 IaaS 服务模式，为用户提供基础设施服务。云平台依托于虚拟化技术将物理资源抽象成虚拟资源池，并为用户提供丰富多样、自动化的管理接口。用户可以借助这些接口实现资源的统一管理，创建操作系统，按需部署应用软件。云平台具有高灵活性和可扩展性，可以根据用户的需求动态调度物理资源，实现虚拟资源的弹性伸缩。由 NASA 和 Rackspace 联合发起的开源项目 Openstack，是一个由控制、网络和计算等若干节点组成的分布式系统，目标是提供实施简单、可大规模扩展、丰富、标准统一的云计算管理平台，默认以虚拟机为媒介提供服务，并为管理虚拟机提供计算、

网络和存储等资源。如图 2-11 所示，Openstack 由一系列低耦合、高内聚的组件构建，每一项组件都为 Openstack 提供一项服务功能，组件之间通过 REST 风格的接口通信和相互调用。Openstack 的核心组件包括计算服务 Nova、镜像服务 Glance、块存储服务 Cinder、网络服务 Neutron 和身份验证服务 Keystone。

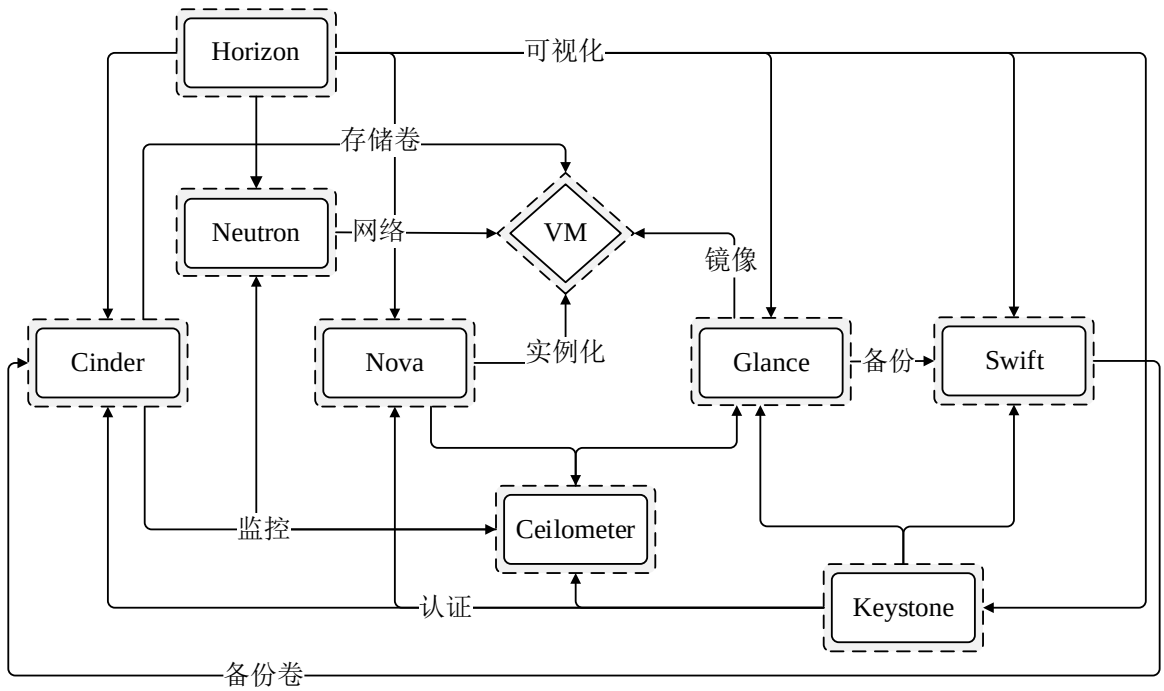

图 2-11 Openstack 架构

Nova 是 Opensatck 最核心的组件，其他服务都围绕 Nova 运行。用户可以通过 Nova 实施创建虚拟机、销毁虚拟机、迁移虚拟机等操作，实现对虚拟机全生命周期的管理。Nova 默认使用 KVM 创建虚拟机，部署的虚拟机具有接近于物理机器的性能。Glance 负责管理各种虚拟机的镜像，用户通过 Glance 可以查询、上传、销毁和下载镜像。这些镜像描述了虚拟机的操作系统和资源信息，通过镜像可以快速恢复虚拟机的运行。Cinder 实现了持久化的存储，负责为虚拟机实例提供磁盘卷。Neutron 组件提供网络资源管理服务，负责在不同虚拟机之间构建网络连接，实现虚拟机之间的互相通信。Openstack 是一个多租户的平台，Keystone 可以用于实现不同用户之间的隔离与通信。用户需要 Keystone 提供的认证令牌 token，才能进入对应的 Project，调用其它组件也需要身份验证，极大地保证了服务的安全性。

2.6 流量回放存在的问题分析

如 2.2 节所述，网络流量仿真的基本目标是生成可以反映网络基本特征和服务能力的流量。相比于数学模型驱动生成流量的方法，流量回放方法可以在最小数据传输单元—流量报文层面，复现流量细粒度特征和服务能力，在逼真性上具有一定优势，已成为主流的仿真方案。因此，本文研究基于流量回放的网络流量仿真方法，具有一定的技术基础和可行性。

对流量回放涉及的相关技术分析可以看出，首先，在流量数据采集方面，XDP 能

从网卡驱动层面操作数据报文，相比于 Libpcap 库更具性能优势，结合 BCC 和 XDP 实现高性能的数据采集技术，为构建流量模板库提供全面、多样化的流量数据，具有合理性和优越性；在流量数据存储方面，以 SPDK 作为模板库存储系统的底层支撑，同时通过 RPC 技术对外界提供存储服务接口，具有可行性与高效性。其次，在流量报文再生成方面，相比于基于内核的报文回放工具，以 DPDK 提供的可扩展框架为基石，构建多样化流量报文回放方法，更具灵活性、扩展性和高效性。最后，选用 Openstack 作为网络流量仿真平台，具有灵活性和扩展性，基于 Openstack Nova 组件可以创建接近真实物理节点性能的流量回放节点，基于 Openstack Neutron 组件可以创建多样化、逼真的网络链路，灵活地结合其他服务可以实现大规模、多样化仿真场景的快速构建。

但为了综合运用上述技术，解决当前互联网流量协议多样、规模庞大、场景复杂等特点所带来的挑战，还需要考虑以下问题：

1）XDP 仅提供了基础的数据捕获功能，可以将流量数据从网卡驱动中抓取出来，但缺乏流量数据过滤和进一步处理的能力。因此，如何过滤符合要求的流量报文，并进一步按需生成可以用于流量回放的多样化流量模板是一个需要解决的问题。此外，有必要结合 SPDK 和 RPC 为仿真节点提供高效的存储服务。

2）DPDK 实现的用户态驱动，具有低延迟和高性能的特点，相比于硬件和 Linux 网络协议栈传输报文的方式，更具灵活性和高效性。然而，尽管 DPDK 为流量报文传输提供了一种新的思路，但缺乏具体的流量回放解决方案。因此，需要考虑合理利用 DPDK 提供的核心组件，设计高性能的流量回放架构，并实现多样化的流量回放方法。

3）OpenStack 可以为流量仿真提供平台支撑，但缺乏实现大规模网络流量仿真的易用方案，在大规模拓扑部署、大规模仿真节点控制方面仍然需要通过手动方式实现，需要研究自动化实现方案，提升仿真的灵活性和易用性。同时，需要考虑仿真实验中的数据分析问题，研究可视化展示仿真任务执行情况的方法。

2.7 本章小结

本章首先阐述了网络流量仿真的目标和相关技术，选取流量回放作为仿真方法的优势和必要性；其次，从流量数据采集与流量数据存储技术、流量报文再生成技术两个方面介绍流量回放的相关技术；再次，对为网络流量回放提供平台支撑的虚拟化与云计算技术进行阐述；最后，分析和总结流量回放相关技术的发展趋势并对应用这些技术存在的问题进行详细分析。

第三章 高性能可扩展流量模板库构建技术

3.1 引言

如前面章节所述，流量回放技术以预先捕获的流量数据作为流量模板，为后续流量再生成提供逼真性保障，目前网络技术高度发展，流量数据的种类、数量也随之剧烈增长，亟须建立流量模板库进行统一的管理。但是在流量模板库构建过程中仍然存在多样化流量模板获取困难和流量模板库存储系统性能不足的问题，这严重阻碍了流量回放技术在各类复杂仿真场景下的应用以及流量回放规模的扩大。针对上述问题，为了实现高效、可扩展的获取流量模板，针对现有基于 Libpcap 的流量数据采集技术在性能上和扩展性上的不足，提出了基于 XDP 的流量数据采集优化技术，旨在高负载网络环境下，以极低的延迟捕获所有符合要求的流量数据，并进一步生成多样化的流量模板；为了解决流量模板库存储性能不足的问题，提出了基于 SPDK 的流量数据存储优化技术，旨在为大规模网络流量回放提供高效的存储服务。

3.2 流量模板库必要性及问题分析

目前大部分的流量回放系统都是面向小规模的仿真场景，流量模板一般存储在回放节点内部，如图 3-1(a)所示。通过这种方式，回放节点能够迅速地将仿真任务需要的流量模板读取到内存中，解析模板后再生成流量数据。但是，随着流量数据和仿真规模的扩大，这种存储方式存在着很大的弊端。首先是灵活性不足，回放节点只能仿真存储在本地的流量协议，每次增加新的流量模板都需要重新构建 KVM 镜像，生成新的仿真节点；其次是资源利用率低，有时某些回放节点会存在重合的流量仿真任务，这种情况下，相同的流量模板会存在冗余的副本，在大规模仿真场景下，容易导致存储资源的浪费。因此，构建一个如图 3-1(b)所示的远程流量模板库是更好的选择，当回放节点需要执行仿真任务时，从远程模板库中拉取对应的流量模板，能极大地节省存储资源，提升流量回放的灵活性。

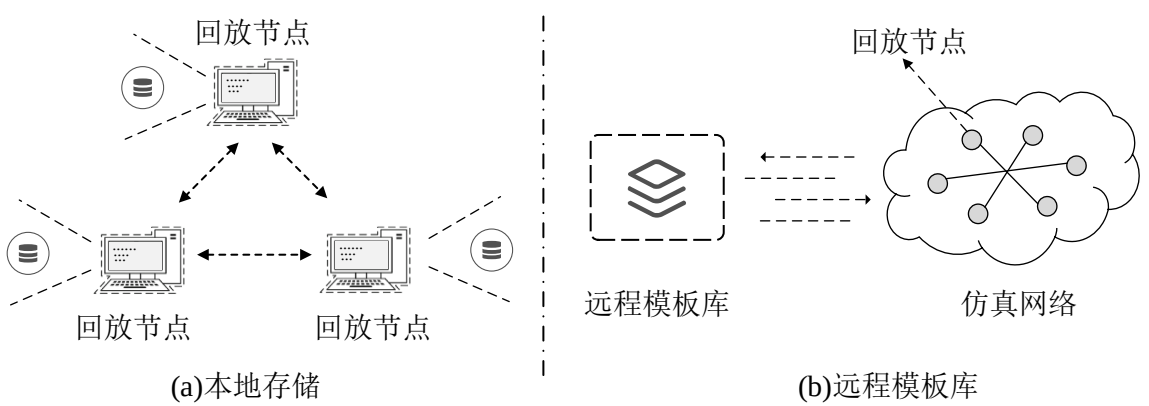

图 3-1 本地存储和远程流量模板库对比

虽然流量模板库可以在一定程度上减少物理存储资源的浪费，提升流量回放的灵活性，但仍然面临两大问题：首先是多样化流量模板的获取，当前网络环境呈现复杂化的趋势，具有流量数据规模庞大、吞吐量高的特点，为了提升协议模板的多样性，

实现对各类现实网络场景的逼真、灵活复现，必须依托于高性能、可扩展的流量数据采集技术；其次是流量模板的存储，不同于本地存储的方式，回放节点从远程流量模板库中读取流量模板时存在一些固有的延迟，在大规模流量回放时，并发度也是一个需要考虑的问题，为了最小化这些延迟，提升仿真规模的可扩展性，必须依托于高效的存储服务。

因此，本章的后续部分将介绍，基于 2.3 节提及的相关技术，为了解决上述问题而提出的流量数据采集优化技术和流量数据存储优化技术。

3.3 基于XDP的流量数据采集优化技术

3.3.1 流量数据采集问题分析

在上一节中对构建流量模板库的必要性进行了分析，并指出了构建流量模板库需要实现高效、可扩展的获取流量模板，以提升协议模板的多样性。为了解决这个问题，必须依托于流量数据采集技术，但目前大部分流量数据采集工具主要应用于网络分析和监测场景，未有针对流量仿真场景适配的流量采集方案。如何从网络流中高效地捕获符合要求的流量数据，为流量模板库提供多样化的流量模板，是一个亟待解决的问题，具体分析如下：

1）目前大部分数据采集工具都是基于 libpcap 库实现，2.3 节对 libpcap 已有介绍。Libpcap 在网卡驱动层的上部、IP 层的底部处理数据包，此时数据包已从原始的数据帧转化为 sk_buff 结构体，产生了额外的消耗。在流量速率较高时，libpcap 的数据包捕获效率会有所下降，延长原始数据包达到应用程序的时间，甚至产生丢包现象，对运行中的业务有一定的影响。

2）现有数据采集工具一般将流量数据直接保存为 pcap 文件或其他特定格式的文件，这些流量数据并不能直接应用于流量回放，还需要手动进行处理以生成符合要求的流量模板，过程繁琐，可扩展性差。

综上所述，现有的数据采集工具在性能和可扩展性上都无法满足为流量模板库提供多样化流量模板的需求。随着 eBPF 技术的兴起，以 eBPF 机制为基础的快速、可编程包处理框架 XDP，通过将钩子函数注入网卡驱动层，能在软件层面最早的节点接触到原始数据帧。此时未进行任何解析包的操作，具有最高的软件性能，在数据包捕获方面存在一定的优势。因此，有必要基于 XDP 设计一个新的数据采集架构，实现自动化捕获符合要求流量数据和进一步生成流量模板的功能，为流量模板库提供多样化流量模板，提升协议模板的多样性。

3.3.2 基于 XDP 的数据采集架构

如图 3-2 所示，本文设计的基于 XDP 的流量数据采集架构可分为两个部分，运行在内核的 XDP 程序和运行在用户空间的 XDP 应用程序。内核 XDP 程序执行真正的捕获操作，拦截网卡驱动层收到的报文，拷贝报文的副本并传递给 XDP APP，然后将原始报文放行给业务程序。网卡在运行过程中会收到大量的流量数据，但有时仿真所需的只是特定的协议流量，如果直接将所有流量数据先拷贝给用户空间的 XDP APP，再

由 XDP APP 筛选符合要求的流量数据，会导致采集效率下降。因为，此时每个数据报文都会存在一份副本，许多 CPU 周期会浪费在无用的数据报文拷贝上，而且额外占据大量的内存。因此，本文提出的架构中，内核 XDP 程序先对数据报文进行过滤操作，仅将少量符合要求的数据报文传递给 XDP APP，以提升报文捕获的效率。XDP APP 收到符合要求的流量数据后，对其进一步处理，生成满足仿真需求的流量模板。

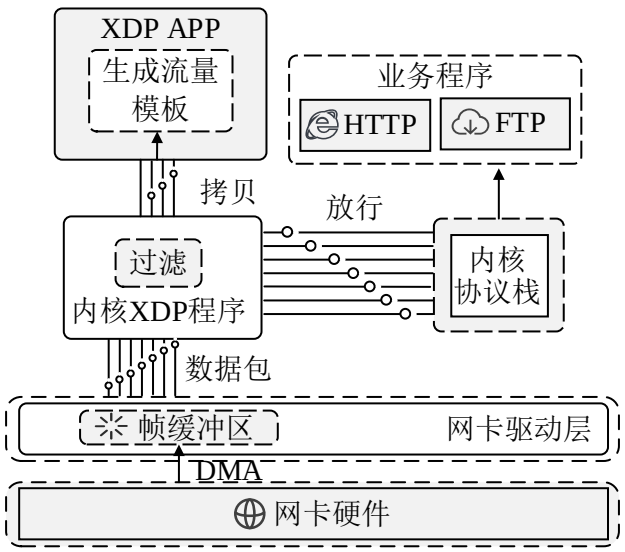

图 3-2 XDP 数据采集架构

本文将于 3.3.3 节和 3.3.4 节对内核 XDP 程序的过滤机制和 XDP APP 生成流量模板的方案进行详细的介绍。

3.3.3 基于 BPF 的流量数据过滤方案

流量数据过滤是指根据报文的协议层字段，包括 IP 地址、传输层协议和端口号等，选择或拒绝符合指定规则的报文。逐层过滤[58-59]是一种有效且易于编程实现的方式，过滤模块从最下层开始解析报文的协议字段，如果不满足用户指定的规则退出过滤模块，放行报文，否则继续向上解析，直至满足所有规则，拷贝该报文的副本至用户空间进行下一步处理。这种方式存在一些弊端，一是不够灵活，只能简单地判断“是”与“否”，如果要支持“或”等其他功能则需要复杂的处理逻辑；二是平均开销大，每条报文都需要逐层判断，容易产生冗余的操作。

虽然 libpcap 库底层的捕获效率低于 XDP，但为了提高自身的易用性，libpcap 设计了一套接近自然语言的过滤规则，并在库中内置了一个小型的编译器，可以将过滤器规则编译为 BPF 字节码，然后将 BPF 字节码注入 Linux 内核并“附着”在指定 Socket 接口。运行在内核的 BPF 虚拟机负责解释这些字节码，协助 Socket 接口对接收到数据包进行判定，执行过滤操作。

然而，XDP 底层采用的是扩展的 BPF 指令集 eBPF，与原始的 BPF 指令集并不兼容，BPF 指令无法在 eBPF 虚拟机上运行，而且 XDP 程序采用 C 语言编写，所以 BPF 过滤器无法直接应用于 XDP 程序中。因此，为了复用 BPF 过滤器，本文设计了如图 3-3 所示数据包过滤架构：

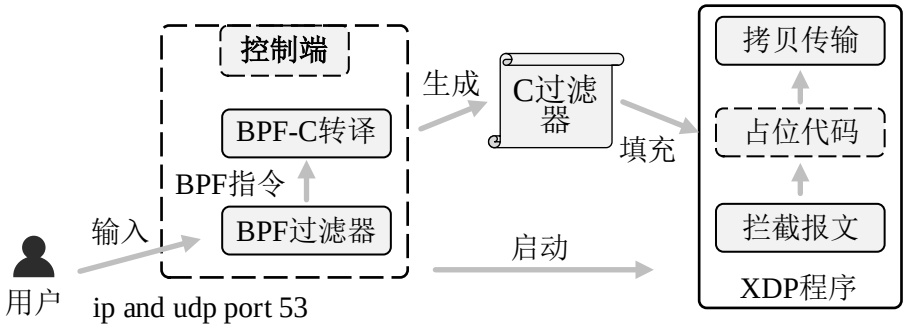

图 3-3 数据包过滤架构

过滤流程可描述为以下步骤：

步骤一：用户输入形如“ip and udp port 53”的过滤规则，启动控制端；

步骤二：控制端解析用户命令，调用 BPF 过滤器，将过滤规则翻译为 BPF 指令，然后将指令输入 BPF-C 转译模块，生成 C 过滤器代码；

步骤三：控制端定位 XDP 程序中的占位代码，用 C 过滤器代码替换这些占位代码，然后启动 XDP 程序，执行过滤动作，拷贝和传输数据报文至用户空间。

同时，为了将符合要求的数据报文高效地传输至用户空间，本节引入 Ringbuf 作为传输的介质，设计了如图 3-4 所示的数据报文消费模型。Ringbuf 是通过内存映射技术申请的一片内存空间，实现了用户进程和内核线程共享同一缓冲区的功能，这使得应用程序可以直接操作内核程序在 Ringbuf 上存储的数据，从而减少系统调用和内存拷贝的开销，降低报文传输的延迟，提升报文处理的速度。

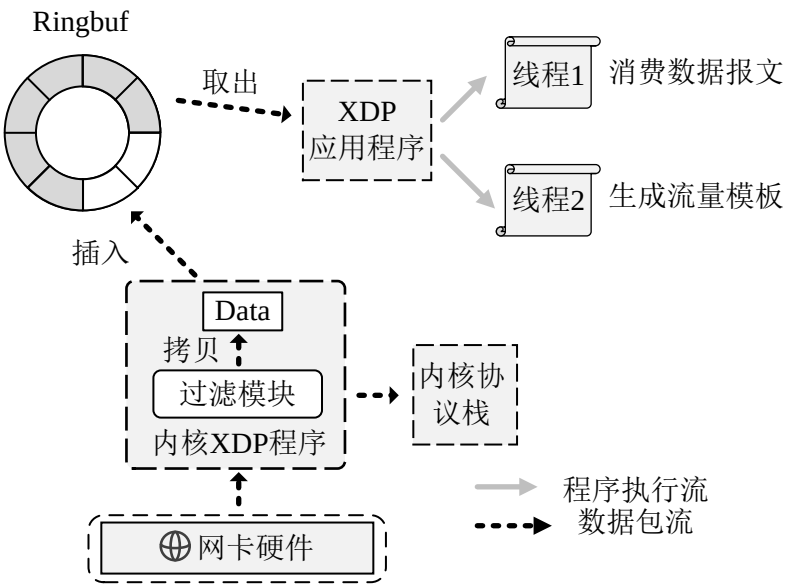

图 3-4 数据报文消费模型

整个模型可分为两部分，内核 XDP 程序生产数据报文和 XDP 应用程序消费数据报文，两个操作并行、异步进行。内核 XDP 程序首先过滤符合要求的数据报文，然后在 Ringbuf 中预先申请内存空间，再构造数据报文副本，插入 Ringbuf 中。XDP 应用程序分为两个线程，线程 1 不断地主动查询 Ringbuf 中是否存在可以消费的元素，将其取出来，交给线程 2，以避免高速流量采集时出现 Ringbuf 缓冲区溢出，产生丢包现象；

线程 2 负责进一步处理采集的数据报文，生成符合流量回放需求的流量模板，此过程将于 3.3.4 节详细介绍。

3.3.4 基于分层解析的流量模板生成方案

流量模板是指包含了原始报文数据和一系列参数的、固定格式的配置文件，通过解析流量模板，可以快速地获取仿真流量信息，灵活地回放真实流量数据，复现多样化的网络场景。捕获的原始流量数据通常并不能直接应用于流量仿真实验，需要实验人员手动进一步地处理，生成所需的应用协议模板。为了提升本文所实现的数据采集框架的扩展性和易用性，适配流量仿真场景，本节介绍自动化生成流量模板的方案，逻辑架构如图 3-5 所示，从下至上分为三层，包括 IP 层解析、传输层解析和应用层解析，通过这三层解析可以提取数据包深层次的信息，将所有数据包按会话流分类，生成细粒度的流量模板。

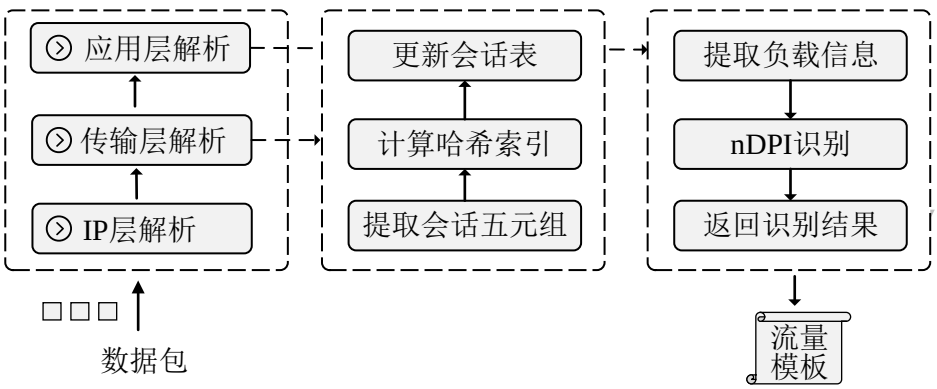

图 3-5 流量模板生成架构

整个流量模板生成的流程可描述为以下步骤：

步骤一：收到捕获的数据包，定位数据包网络层的位置，解析 IP 报文头部，得到源 IP 地址、目的 IP 地址、传输层协议号(TCP/UDP)和 IP 首部的长度。

步骤二：根据 IP 首部长度，计算传输层的偏移量，解析数据包传输层的信息，得到源端口信息和目的端口信息，根据公式 3-1 计算五元组的哈希值，通过哈希索引查询会话表中是否存在对应的会话流。如果会话流已存在，则更新会话流的统计信息，否则，在会话表中新建一个表项，用于统计该会话流后续数据包信息。

$$index = (src\_ip + dst\_ip + src\_port + dst\_port + l3\_pro)\%Num \tag{3-1}$$

其中 $index$ 为会话表索引，src_ip 为数据包源 IP，dst_ip 为数据包目的 IP，src_port 为数据包源端口号，dst_port 为数据包目的端口号，$l3\_pro$ 为传输层协议号，Num 为会话表大小。

步骤三：提取数据报文负载，准备识别应用层协议信息，对数据报文进行分类。一般而言，一些常见的应用层协议有着固定的端口号，如 HTTP 协议使用 80 号端口、Telnet 协议使用 23 号端口。但是，为了防止不法分子窃取私密信息，近些年来越来越多的应用层协议使用了可变端口号。因此，常规的五元组分类方法在当前网络环境受到了极大地限制。为了解决这个问题，本节引入 nDPI[60]识别应用层的负载信息。nDPI 采用深度报文检测技术，可以提取数据包的负载特征，并将之与特征数据库进行匹配

来识别应用层协议。

步骤四：继续处理下一个数据包。在所有数据包处理完毕后，根据分类结果记录每条流的信息，导出为 YAML 格式的应用协议流量模板。

为了保存应用协议的信息，方便流量回放时实验人员对会话流进行宏观层面的控制，本文设计了如表 3-1 所示的基础流量模板。流量模板由后端原始流量文件和前端模板参数组成，原始流量文件的格式为 pcap，同一条流中的所有数据包都按序保存在其中；模板参数包含了一些控制参数，如回放持续时间、IP/端口分布方式等。流量仿真时，回放节点解析原始流量文件，根据控制参数依次重构和发送数据报文。例如，当 IP 分布方式设置为递增时，就在回放流量时以递增的方式循环修改每个数据包的 IP 地址。

表 3-1 基础流量模板

字段	类型	描述
name	string	模板名称
duration	float	回放最长持续时间，默认为 0，表示回放完所有数据包
distribution	string	IP/port 分布方式，有随机和递增两种方式
src_ip_start/end	string	数据包源 IP 地址起始/结束值，默认为流的源 IP 地址
dst_ip_start/end	string	数据包目的 IP 地址起始/结束值，默认为流的目的 IP 地
src_port_start/end	string	数据包源端口起始/结束值，默认为流的源端口
dst_port_start/end	string	数据包目的端口起始/结束值，默认为流的目的端口
traffic_file	string	原始流量文件路径

此外，为了满足不同的仿真需求，可以对基础流量模板的配置参数进行删改，后续只需要回放时针对不同的流量模板采取不同的解析方式即可。

3.4 基于SPDK的流量数据存储优化技术

3.4.1 流量数据存储问题分析

建立流量模板库可以实现对流量模板的统一管理，优化和最大化物理存储资源的利用率。但不同于本地存储，从远程流量模板库中读取流量模板时存在一定的延迟，在大规模网络流量仿真时，模板库的并发度也是一个需要考虑的问题。在本文的流量仿真环境中，网络延迟一般比较低且相对稳定，所以网络延迟不在本节的讨论范围内。对于流量模板库自身的存储系统而言，性能瓶颈主要存在以下两方面：

1）硬件瓶颈。传统方法中，流量数据以机械硬盘作为存储介质，机械硬盘虽然价格实惠，但其硬件性能较差，流量数据从机械硬盘传输到内存中的速度较慢、延迟较高。

2）软件瓶颈。用户程序在读取流量数据时首先需要陷入内核，触发系统调用，然后经过虚拟文件系统、内核文件系统的调度，最终从本地磁盘设备获取所需资源，整个过程十分漫长。

因此，有必要从硬件和软件两方面提升流量模板库的存储性能，以提升仿真规模的扩展性。硬件方面，近年来，NVMe 固态硬盘飞速发展，相较于机械硬盘，存储性

能有了极大地提升，在并发度和延迟上都能满足大规模流量仿真的需求。软件方面，Intel 提出的 SPDK，实现了高性能的用户态驱动，可以将整个存储过程由内核迁移到用户态，消除系统进入内核带来的开销，相比于原有的内核驱动，能提供更低且更稳定的延迟，可以充分发挥出 NVMe 固态硬盘的硬件性能。此外，SPDK 还提供了一个基于事件驱动的可伸缩、高性能的编程框架，可以方便开发人员在 SPDK 的基础上构建高效的存储服务，具有良好的扩展性。

综上所述，有必要以 NVMe 固态硬盘为硬件支撑，在 SPDK 的基础上构建高性能存储架构，从而提升流量模板库的存储性能，向仿真节点提供高效的存储服务，以提升仿真规模的可扩展性。

3.4.2 基于 SPDK 的异步 RPC 模型

由于远程过程调用(RPC)具有低耦合、易于扩展的特性，当前大多数分布式系统的组件之间都是通过 RPC 进行网络通信，因此本文也基于 RPC 对外提供存储服务的接口。基于上述分析，本文以 NVMe 固态硬盘为硬件支撑，在 SPDK 框架的基础上构建了一个基于轮询驱动的异步 RPC 模型，以向仿真节点提供高效的存储服务，具体架构如图 3-6 所示：

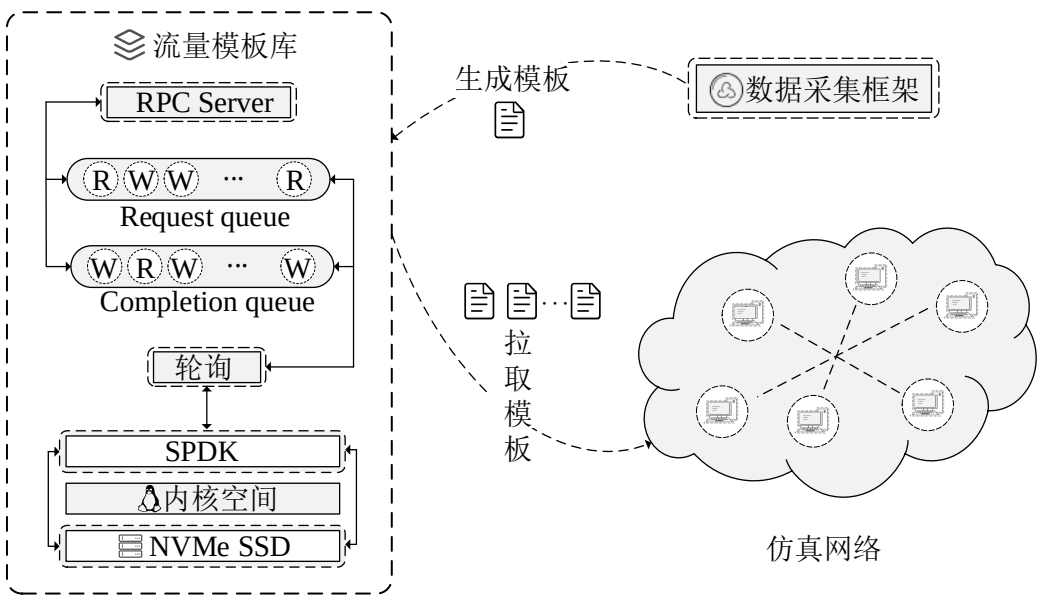

图 3-6 存储服务架构

首先，无论是请求还是响应，所有的 RPC 消息都采用统一的格式封装。为了充分利用现代 CPU 的缓存机制，本文设计了如表 3-2 所示 RPC 消息头部格式，可以储存在 64 位 CPU 的高速缓存行中，方便 CPU 快速地获取 RPC 消息的详细信息。

表 3-2 RPC 消息格式

字段	位数	描述
flags	8 位	标签，标记 RPC 消息是请求还是响应
req_num	8 位	请求号，指出本次调用的函数
seq	8 位	序号，RPC 消息的唯一标识

mes_len	16 位	消息长度，指明 RPC 消息的大小，必须对齐高速缓存行
ret	16 位	返回 RPC 的处理状态

当模板库的 RPC 服务端接收到调用请求后，会对 RPC 请求进行解封装，判断是读取模板或是在库中写入模板，然后重新将请求参数封装成可以被 SPDK BlobFS 接收的数据格式。在同步等待的处理方式中，接下来 RPC 服务端会将请求转发给 SPDK BlobFS，等待下层处理完成后，将处理完成的响应消息重新封装为 RPC 格式发送出去。但下层处理请求时，RPC 服务端处于阻塞状态，在高并发场景下，新到的请求无法及时处理，导致响应延迟。

因此，为了避免同步等待带来的阻塞开销，本节采用异步回调的方式处理请求，在 RPC 服务端和 SPDK 框架中封装一层缓存队列，包括 Request queue 和 Completion queue。Request queue 和 Completion queue 底层以无锁环形队列的形式实现，开销极小。此时，RPC 服务端解析 RPC 请求，重新封装请求参数，将其加入 Request queue 队列首部后，不再等待下层处理完成，而是接着处理下一个请求。轮询模块不断查询 Request queue，从队列尾部取出请求，通知 SPDK BlobFS 从固态硬盘中读取或者写入文件。请求处理完成后，将下层返回的响应数据进行封装，加入 Completion queue，并以回调的方式通知 RPC 服务端，包装成 RPC 响应消息并发送给请求方。

3.5 实验分析与验证

本节对基于 XDP 的数据采集优化技术和基于 SPDK 的数据存储优化技术进行验证，包括流量数据采集功能验证、流量数据采集性能对比和模板库存储服务性能对比。

3.5.1 实验环境

验证数据采集功能和性能的基础实验平台由四台戴尔 R730 服务器组成，在上面搭建了 Mitaka 版本的 Openstack 云平台，由控制节点、网络节点和计算节点组成，具体配置如表 3-3 所示。在云平台上创建的虚拟节点配置均为 4vCPU、8GB 内存、128GB 硬盘，操作系统为 Ubuntu22.04。

表 3-3 Openstack 配置表

配置	控制节点	网络节点	计算节点 1	计算节点 2
型号	Dell R730	Dell R730	Dell R730	Dell R730
CPU	Intel E5-2609 v4*2	Intel E5-2620 v4*2	Intel E5-2620 v4*2	Intel E5-2620 v4*2
内存	64GB	128GB	128GB	128GB
硬盘	1TB	1TB	2TB	2TB

由于戴尔 R730 服务器与高性能的 NVMe 固态硬盘不兼容，因此本文的流量模板库构建在一台联想小新 Pro16 2022 笔记本上，笔记本处理器为 AMD 6800H 标压版本，内存为 16GB，搭载 512GB NVMe 固态硬盘，操作系统为 Ubuntu22.04。后续模板库将通过虚实互联技术与构建在云平台上的网络流量回放仿真系统集成。

3.5.2 流量数据采集功能验证

本文提出的流量数据采集优化技术，主要目的是从网络流中高效地捕获符合要求

的流量数据，为流量模板库提供多样化的流量模板。因此本节构建了如图 3-7 的实验场景，对流量数据采集功能进行验证，包括流量数据过滤和生成流量模板的功能。

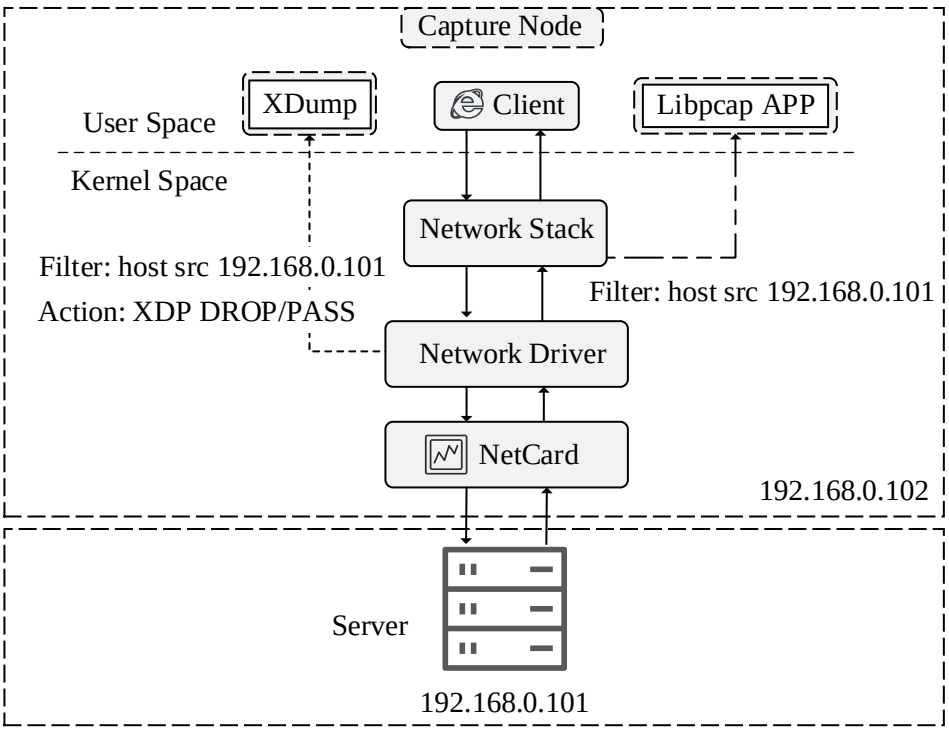

图 3-7 数据采集功能验证场景

实验场景包括两个节点，均创建在 Openstack 云平台上，操作系统为 Ubuntu22.04，Linux 内核版本为 5.17，满足 XDP 程序的运行环境。其中一个节点是捕获节点 Capture Node，IP 地址为 192.168.0.102，在上面运行着三个应用程序，包括本章设计的数据采集框架 XDump、基于 libpcap 库实现的数据采集程序 libpcap APP 和用于发送请求的 Web 浏览器客户端 Client；另外一个节点是服务器节点 Server Node，IP 为 192.168.0.101，Server Node 上运行着一个 Web 服务器程序，用于回应 Cilent 发送的请求，产生通信的流量数据。

实验开始时，首先，启动 XDump，设置过滤表达式为“host src 192.168.0.101”，含义为捕获源 IP 为 192.168.0.101 的报文，即捕获 Server Node 发送给 Capture Node 的报文，然后设置捕获报文后的动作为 XDP DROP，即丢弃该报文的原始帧，不将其传递给内核协议栈。其次，启动 Libpcap APP，设置过滤表达式为“host src 192.168.0.101”。最后，启动 Client，向 Server Node 发送正常的请求，进行交互式通信。

XDump 捕获的报文数据如图 3-8 所示，包含了 Server Node 重传的 3 个 SYN-ACK 报文，与设置的过滤规则匹配，说明 XDump 能实现数据过滤的功能。

Source	Destination	Proto	Leng	Info
192.168.0.101	192.168.0.102	TCP	78	41668 → 53408 [SYN, ACK] Seq=0 Ack=1 Win=819
192.168.0.101	192.168.0.102	TCP	78	[TCP Out-Of-Order] 41668 → 53408 [SYN, ACK]
192.168.0.101	192.168.0.102	TCP	78	[TCP Out-Of-Order] 41668 → 53408 [SYN, ACK]

图 3-8 捕获报文分析

而 Libpcap APP 捕获的数据包个数为 0，这是因为，XDump 内核程序的 HOOK 点在网络驱动程序中，可以直接操作原始的数据帧，匹配到符合规则的报文之后，复制

一份报文副本，传递到用户空间的 XDump 程序，然后根据设置的 XDP DROP 动作，将报文的原始帧丢弃。Libpcap APP 的程序 HOOK 点在内核协议栈底部，无法获取 XDump 丢弃的原始数据帧，自然捕获不到匹配过滤规则的报文数据。同样，依靠内核协议栈通信的 Client 也收不到 Server Node 发送的响应报文，无法根据响应报文对 Server Node 发送进一步的请求，而 Server Node 以为响应报文丢失，不断重传响应报文。

为了验证 XDump 生成流量模板的功能，再次进行实验，在 Server Node 上以 Capture Node 为目标，回放 CAIDA[61]提供的流量数据集，该数据集包含互联网中的各种应用流量。同时，将 XDump 捕获报文后的动作设置为 XDP PASS，过滤规则设置为捕获所有 IP 报文，其他设置与上次实验保持一致。此次 XDump 捕获报文的统计数据如表 3-4 所示：

表 3-4 捕获的报文统计表

属性	值	描述
cap_pkts	791615	捕获的数据包数量
drop_pkts	0	丢弃的数据包数量
total_pkts	791615	经过 XDump 的数据包总数
cap_bytes	355417784	捕获的数据包字节数
drop_bytes	0	丢弃的数据包字节数
total_bytes	355417784	经过 XDump 的字节总数

基于捕获到的流量数据识别出的应用协议和流的数量如上图 3-9 所示，横坐标为捕获的数据包个数，主纵坐标为识别出的应用协议数量，副纵坐标轴为识别出的流数量。可以看出，大部分的应用协议和流在前 100K 个数据包内被识别出来，随着捕获报文数的增加，应用协议和流的识别数分别达到了 132 种和 40386 条。最终，一共生成了 40386 个流级别的流量模板。

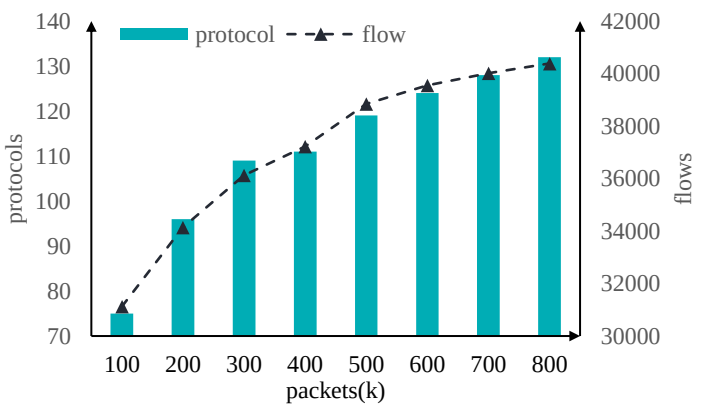

图 3-9 应用协议和流识别数量

3.5.3 流量数据采集性能对比

目前大部分数据采集工具均基于 Libpcap[41]实现，为了验证本文提出的基于 XDP 的数据采集技术(XDump)在性能上的优势，本节分别选用 Libpcap 和文献[59]提出的 NPCA(Network Packet Capture and Analysis System Based on eBPF)与 XDump 进行对比，验证采集数据包的时延和丢包率。实验方法为，在云平台上创建两个节点，一个为发