



专题：智能通信技术的应用

高性能网络协议栈兼容性研究

姜惠友, 李峻峰, 李丹
(清华大学, 北京 100084)

摘要: 人工智能系统的海量数据处理需要很高的网络吞吐来支撑, 网络协议栈作为上层人工智能应用和底层高速硬件网卡之间的桥梁, 对整个系统的网络性能起到关键性作用。通过对主流网络协议栈兼容性进行详细对比分析, 发现目前各类协议栈兼容性普遍较低, 导致人工智能应用移植困难等问题。提出了高性能用户态网络协议栈的高兼容性设计构想, 使各类传统应用无需修改源码即可获得网络性能的提升, 讨论了高性能协议栈的后续发展方向。

关键词: 兼容性; 网络协议栈; 高性能

中图分类号: TP393

文献标识码: A

doi: 10.11959/j.issn.1000-0801.2019096

Research on compatibility of high-performance network stack

JIANG Huiyou, LI Junfeng, LI Dan
Tsinghua University, Beijing 100084, China

Abstract: The massive data processing in artificial intelligence system requires the support of high-throughput network. As a bridge between the upper artificial intelligence applications and the underlying high-speed hardware network adapter, the network stack plays a crucial role in the performance of the whole network system. Through detailed analysis and comparison of the compatibility of mainstream network stacks, it is found that the compatibility of various network stacks is generally low, which leads to difficulties in transplantation of artificial intelligence applications. The idea of achieving both high-compatibility and high-performance of user-level network stack were proposed. In this way, legacy applications could directly attain the improvement of performance without modification of the source code. Path for the subsequent development of network stack was lit up.

Key words: compatibility, network stack, high-performance

1 引言

现代的高性能人工智能处理系统需要端系统的高速支持, 而网络是高性能服务器系统的性能

瓶颈, 因此高性能用户态协议栈对人工智能处理系统至关重要。互联网早期网络环境简单, 设备带宽资源有限, 基于尽力服务 (best-effort) 原则的网络协议栈足够满足当时的需求。然而移动互

收稿日期: 2019-02-12; 修回日期: 2019-05-06

基金项目: 国家自然科学基金资助项目 (No.61772305, No.61432002)

Foundation Items: The National Natural Science Foundation of China (No.61772305, No.61432002)



联网和人工智能的快速发展使得数据中心的网络流量飞速暴涨，HTTP、RPC 等常见通信协议存在着的大量 TCP 短连接让网络协议栈不堪重负。经过实验证明^[1]，在 8 个节点上通过 TensorFlow 对典型的人工神经网络 VGG19 进行分布式训练，90% 的等待时间消耗在网络传输上。与此同时，40 GB、100 GB 带宽的高性能网卡已逐渐部署，硬件设备的快速发展使得高性能人工智能服务器系统性能瓶颈的压力转移到网络协议栈上。

传统 Linux 内核网络协议栈进行了拆分 Accept 共享队列^[2]、避免“惊群”^[3]、零复制^[4]等优化。Affinity-Accept^[5]基于 2.6.35 版本的 Linux 内核通过 Accept 队列本地化、减小连接 Hash 映射表锁的粒度等优化保证一个 TCP 连接全过程只由一个 CPU 核处理。MegaPipe^[6]通过在内核网络 API 与网络应用之间建立系统调用批处理通道，最大程度降低网络系统调用的上下文切换开销，并将虚拟文件系统 VFS 从网络 Socket 中剔除，设计出更轻量级的 Socket API。FastSocket^[7]除了共享资源拆分、连接本地化与 VFS 优化之外，还实现被动连接和主动连接的本地化，避免 TCP 数据在 CPU 之间跨核 Cache Miss 的开销。然而这些在内核进行协议栈优化带来的性能提升较为有限。

本文重点对用户态高性能网络协议栈及其兼容性进行对比分析，并提出基于用户态网络协议栈和高速 I/O 接收分组模式实现高度兼容性的设想，使得当前主流网络应用（如 Nginx、Redis、人工智能端系统等）在完全不需要修改源码的情况下直接享受协议栈加速带来的收益。

2 高性能用户态网络协议栈的研究

网络协议栈自诞生起就属于内核操作系统的一部分，主要是为了保证较高的安全性和性能，却具有难以调试、迭代周期长等缺点。虽然自 20 世纪 80 年代起研究人员不断尝试解决这个问题，将协议栈搬离到用户态来实现^[8-10]，但都无法取得优

于内核协议栈的网络性能，其根本原因是当时还没有一种绕过内核直接完成网络收发分组的高性能技术，导致在用户态进行分组解复用（packet demultiplex）的上下文开销过大^[11]。DPDK^[12]、Netmap^[13]、PF_RING ZC^[14]等高速 I/O 收发分组框架的出现使得分组解复用完全可以绕过内核进行，也让用户态实现高性能网络协议栈变得可行。

在具备高效的绕过内核（kernel bypass）技术后，用户态协议栈最近几年在学术界和工业界都得到飞速发展。mTCP^[15]是近些年学术界影响力较大并已开源的用户态协议栈，除了连接本地化、共享资源拆分等常见协议栈优化之外，mTCP 从底层 I/O 接收分组到用户态网络 API 都采用批处理操作，并且针对短连接采用基于优先级队列的报文处理优化策略，提升短连接流量下多核系统的性能及可扩展性。其系统架构如图 1 所示，底层用户态高速网络接收分组模块采用模块化的设计，可灵活支持 PSIO、DPDK、Netmap 等模块，每一个 CPU 核拥有独立专享的 TCP 监听表、建连表等资源，避免锁竞争同步开销。网络应用进程绑定在唯一的 CPU 核上，该进程包含一个 mTCP 协议栈线程和一个网络应用线程，前者负责网络数据分组的收发以及 TCP/IP 网络协议逻辑处理，后者负责网络应用代码执行逻辑，两者通过队列来完成线程间消息传递通信。然而，相同 CPU 核上的两个线程可能发生严重的 CPU 资源争抢，对此 mTCP 通过 pthread 条件变量来完成两个线程的调度。比如在 mTCP 协议栈线程完成一轮数据分组的接收后产生新建连接、数据分组接收等多个事件（event），并将其放入 event 队列后，便会调用 pthread_cond_signal 去唤醒网络应用线程在 epoll_wait 函数阻塞的执行流，而 mTCP 协议栈线程将会被挂起。mTCP 通过这种主动让出 CPU 的调度策略最大程度地减少 CPU 资源竞争。但是线程调度具有上下文切换开销，对此 mTCP 通过对报文接收、event 通知进行批处理操作来均摊开销。

实验结果显示,在8核系统中 mTCP 比传统 Linux 网络协议栈性能提升 25 倍。

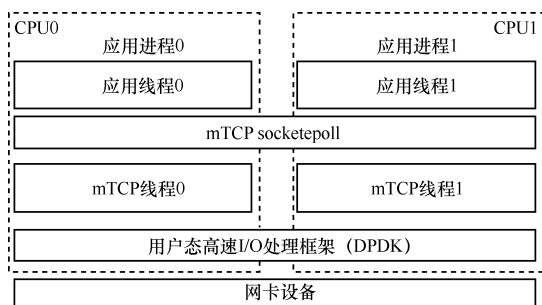


图1 mTCP 系统架构

IX^[16]采用数据平面与控制平面分离的策略,控制平面负责对 CPU 和内存资源配置、申请、调度和监控,数据平面负责网络协议栈的逻辑处理,并利用 I/O 虚拟化技术将数据平面不同网络应用进行隔离,以保证安全性。IX 数据平面的接收分组是在特定网卡上基于轮询 I/O 模式进行的,这样可以减轻中断接收分组模式在高吞吐网络情况下频繁的硬件中断对 CPU 造成的压力。IX 在一个线程中采用 run-to-completion 的工作模式,该模式的完整过程如图 2 所示,包含虚拟网卡接收分组、协议栈拆分组、交付网络应用、应用返回响应数据、协议栈封装分组、网卡发送分组等过程。同一批数据分组整个过程处理完才进行下一批数据分组的处理,以减少系统中不必要的缓存队列,从而最大化降低网络时延。此外 IX 对收发分组和系统调用进行动态调整的批处理,灵活应对不同网络情况,并均摊上下文切换开销。

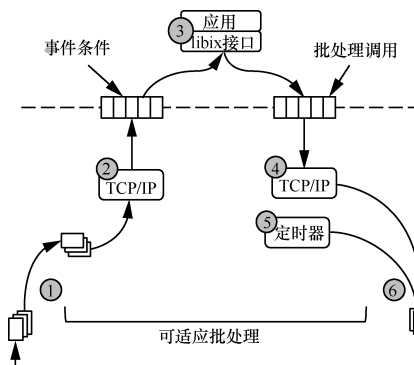


图2 IX run-to-completion 模型

Zygos^[17]在 IX 基础上解决网络执行流在不同核之间的调度问题,从而实现微秒级时延并满足超低尾时延指标的网络性能要求。Arrakis^[18]同样是采用将数据平面从内核中剥离的策略,并利用 I/O 虚拟化和内存映射等技术,使得网络应用可以直接操作虚拟 I/O 设备从而绕过内核。实验结果显示其获得了数倍的网络性能提升。

MultiStack^[19]与 StackMap^[20]在高性能用户态协议栈与兼容传统网络应用之间采取折中策略。两者的核心思想都是将用户态协议栈与传统内核协议栈进行整合。MultiStack 具体实现如图 3 所示,其最底层使用 I/O 虚拟化技术进行分组解复用,向上承载着用户态协议栈和内核协议栈两种系统,用户态路径对应的虚拟网卡使用 Netmap 作为收发分组框架,内核态路径对应的虚拟网卡使用传统基于中断的接收分组模式。StackMap 在设计上采用数据通道和控制通道分离的思想,数据通道通过 Netmap 高性能收发分组模块来减少数据复制。

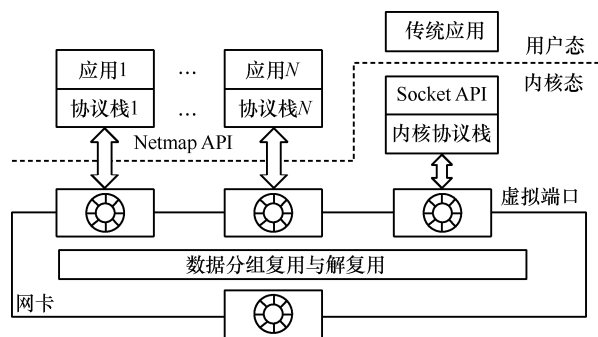


图3 MultiStack 系统架构

工业界近几年也产出了众多优秀的用户态协议栈^[21,24]去提升企业网络服务性能,其中腾讯推出的 F-stack^[23]基于 DPDK 高速 I/O 收发分组框架并移植 FreeBSD 网络协议栈来实现加速网络服务,并已经在 DNS 服务器等实际生成环境中落地。Seastar^[24]是基于事件驱动、纯异步编程的用户态协议栈,利用 futures 和 promises 编程模型将网络 I/O、磁盘 I/O 等统一抽象为异步事件进行高效处



理，其已用于加速高性能 NoSQL 数据库。

3 网络协议栈兼容性分析与对比

用户态网络协议栈由于利用高性能 I/O 收发分组框架或虚拟化 I/O 设备绕过内核从而获得显著的网络性能提升，但是其普遍存在兼容性较低、难以移植传统应用的问题，本节对这些主流的网络协议栈进行兼容性的分析与对比。

3.1 网络协议栈兼容性分析

mTCP^[15]独立开发的简易 TCP 网络协议栈，其无法支持 UDP 应用并难以应对常见的网络攻击。其向上以类似 POSIX 语义支持常见的网络 Socket API 以及 epoll 相关事件接口，虽然参考文献[15]指出只需要对网络应用进行少量源码修改即可完成应用的移植，但从其开源至今 4 年多时间里，mTCP 也仅支持了 Lighttpd 一个较为主流的网络应用服务器，虽有很多开发人员对 Nginx 进行了多次移植尝试^[25]，但经过实验验证，均无法正常运行或者性能不及 Linux 内核协议栈。IX^[16]封装成 libix 库向上层应用提供 API，包括 connect、accept、sendv、recv_done、close 等接口，虽然函数名与 POSIX 网络接口相同，但由于 IX 向应用暴露了流量控制条件信息，造成 libixAPI 的语义发生改变，例如 sendv 函数返回值不再是协议栈缓存的数据字节数，而是协议栈已经成功发送的数据字节数。Event Conditions 是由协议栈数据平面产生并传递给网络应用的相关事件，包括 recv、sent 等事件告知应用数据成功被接收或发送。为了追求更高的性能，IX 既没有遵从 POSIX API 的语义，也没有实现 epoll 相关常用的网络 API，因此具有较低的兼容性，使当今主流网络服务器应用 Nginx、Lighttpd 等难以移植到 IX 上。

MultiStack^[19]与 StackMap^[20]采用较为折中的设计，针对传统网络应用使用内核协议栈的系统调用，而为高性能应用提供一套新的网络 API。该设计仅实现了既可以运行传统网络应用又能运

行新开发的高性能网络应用的目标。然而当下虚拟化技术已很成熟，创建新的虚拟机来运行高性能网络应用的成本很低，所以该折中的设计思想意义并不大，况且传统网络应用若想获得用户态协议栈带来的加速依然需要根据新 API 语义进行重新编写，并没有从根本上解决兼容性问题。

Arrakis^[18]相比之下在兼容性方面做了更多考虑，给网络应用暴露 Native Arrakis API 和 POSIX API 两种接口，前者由于重新设计数据 I/O 的接口而实现了真正的数据零复制，后者由于 POSIX API 中 recv、send 等收发函数中带有应用 buffer 的语义而无法实现真正的零复制，但也获得相比传统 Linux 内核协议栈在网络性能上的提升。

3.2 网络协议栈兼容性对比

用户态高性能网络协议栈 mTCP^[15]由于移植网络应用需修改其源码导致至今只成功应用在 Lighttpd 一款较有影响力的网络服务器应用中，内核态网络协议栈 FastSocket^[7]因具备高度应用兼容性已部署到新浪微博的生成环境中。不难看出兼容性对高性能网络协议栈的后续发展至关重要。本文总结出高性能网络协议栈的兼容性主要体现在以下几个方面。

首先，协议栈是否支持 POSIX API，传统的网络应用大多是基于 Socket、Bind、Listen 等这套经典的 POSIX 网络 API 进行开发，所以是否直接支持 POSIX API 或者接口设计上是否类似 POSIX API 在很大程度上决定了高性能协议栈移植网络应用的开发成本。

其次，是否需要修改网络应用源码是衡量协议栈是否具备高度应用兼容性的重要指标，基于用户态协议栈实现网络应用移植的零源码修改既需要协议栈接口在设计上与 POSIX API 完全一样，又需要对原操作系统网络调用 API 进行劫持，还需要考虑劫持的网络 API 对非网络文件描述符的支持等问题，存在较高的技术挑战性。

最后，协议栈支持的高并发网络编程模型同

样是影响兼容性的关键因素。通常网络 Socket 的读写操作都是同步的,即代码执行到 `recv`、`send` 等网络 I/O 函数时会阻塞并被操作系统挂起等待 I/O 结果返回。Nginx、Lighttpd、Redis 等各种网络应用都利用 I/O 多路复用 (Epoll I/O) 实现高并发网络连接处理,所以支持 Epoll 高并发编程接口是实现移植传统应用不修改源码的必要条件。实现高并发网络连接处理的技术除了 Epoll I/O 多路复用,还有多线程、异步 I/O 模式、协程机制等,但在当前主流网络应用中应用较少。

本文将以上 3 点作为衡量高性能网络协议栈兼容性的主要指标,对主流协议栈进行对比分析,并同时列举了各种协议栈具体支持的网络 API。见表 1,其中 Affinity-Accept^[5]、MegaPipe^[6]、FastSocket^[7]是在内核态对协议栈进行共享 TCP 连接资源的拆分、虚拟文件系统的剔除、连接本地化等优化,但由于无法避免内核系统调用的开销以及无法利用高性能 I/O 收发分组模块,这些内核态协议栈的性能提升往往比较有限,但直接对 Linux 内核网络协议栈进行调优有利于实现协议栈的高度兼容,大大降低网络应用的移植成本。mTCP^[15]、IX^[16]、F-stack^[23]、Seastar^[24]均是在用户态基于高性能收发分组框架实现的协议栈,在获得大幅性能提升的同时却难以兼顾对网络应用的兼容。

4 用户态实现高兼容性的设想

用户态协议栈可以取得显著的网络性能提升,且具有编程空间大、灵活度高、接口易于向 POSIX API 语义靠拢等优势,因此本文提出在用户态实现兼具高性能和高度兼容性的网络协议栈。用户态协议栈具体实现分为协议栈与网络应用共用 CPU 核与分核两种设计思路,前者如 mTCP 需要进行协议栈执行流与应用执行流的合理调度,这样带来的上下文切换开销还需用批处理等其他优化手段进行均摊,然而这种策略对

CPU 资源的利用率较高;后者如 IsoStack^[26]将协议栈用专用核来实现,减少上下文切换的开销,但同时也带来数据在不同核直接迁移造成的 CPU 缓冲不命中等开销。

此外,本文在用户态协议栈实现高度兼容性进行如下几点初步设想,为其该领域的研究提供新思路。

(1) 对 POSIX API 的接管方面需要将协议栈接口部分编译成动态链接 so 库,并利用 LD_PRELOAD 环境变量对网络应用动态链接库加载顺序进行修改,即用户态协议栈提供网络 API 的动态链接库优先于 POSIX 网络系统调用的 Glibc 库进行加载,从而完成对操作系统内核网络接口的劫持。

(2) 由于 `read`、`write`、`epoll` 等函数的文件描述符可能不是 Socket 网络 fd (文件描述符),比如传统应用调用 `read` 对磁盘文件描述符进行读操作,这迫使用户态协议栈系统需要将网络 fd 与非网络 fd 进行区分管理,对网络 fd 进行协议栈的处理流程,对非网络 fd 需要间接调用内核系统接口。因为文件描述符为有符号型整数,较为简单的设计是将正整数空间以 2^{30} 为阈值一分为二,并在所有涉及网络文件描述符的函数入口和返回值进行相应映射,若 fd 大于阈值即为用户态协议栈管理的 Socket 网络 fd。然而 Redis、Lighttpd 等网络应用将 Socket fd 作为数组下标使用,上述 fd 空间管理方式必然导致数组越界访问从而引发系统崩溃,所以需设计一种动态映射方式来管理 fd 资源。

(3) Nginx、Lighttpd、Memcached 等应用在高并发网络环境下会使用多进程或多线程模式进行多核扩展,这便需要用户态协议栈接管 `fork` 和 `pthread` 相关接口,并在并发流创建时能将其绑定在合适的 CPU 核上,防止操作系统的调度机制使多个网络应用执行流争抢同一 CPU 资源从而造成性能下降。此外,由于 `fork` 创建进程时,子进程会对父进程的地址空间进行写时复制



表 1 高性能协议栈兼容性对比

协议栈名称	是否支持 POSIX API	是否需要修改 应用源代码	高并发网络 编程模型	具体已支持的 API
Affinity- Accept ^[5]	是	不需要	Epoll	Linux 2.6.35 版本网络相关 POSIX API
MegaPipe ^[6]	否	需要	事件注册 分发	基于事件注册的接口 mp_create、mp_register、mp_unregister、mp_accept、mp_dispatch、mp_read、mp_write 等
FastSocket ^[7]	是	不需要	Epoll	Linux 2.6.32 版本网络相关 POSIX API
mTCP ^[15]	POSIX-Like	需要	Epoll	mtcp_Socket、mtcp_bind、mtcp_listen、mtcp_accept、mtcp_connect、mtcp_read、mtcp_write 等通用网络 API 以及 mtcp_epoll_wait、mtcp_epoll_ctl 等 epoll 相关接口
IX ^[16]	否	需要	重定义网络事件、弹性线程	批处理优化但语义改变的接口 connect、accept、sendv、recv_done、close 以及数据平面事件响应接口 knock、connected、recv、sent、dead
F-stack ^[23]	POSIX-Like	需要	Epoll、Kqueue	ff_Socket、ff_bind、ff_listen、ff_accept、ff_connect、ff_shutdown、ff_send、ff_writev、ff_sendto、ff_sendmsg、ff_dup 等更丰富且同时支持 TCP 与 UDP 的通用网络 API 以及 ff_select、ff_poll、ff_kqueue、ff_epoll_wait 等多种 I/O 多路复用机制接口
Seastar ^[24]	否	无兼容性	异步 I/O	事件驱动、非阻塞基于 future 的纯异步编程框架，seastar::make_ready_future、seastar::future、seastar::sleep().then()等

(copy-on-write)，即子进程可以调用 close 关闭父进程的 Socket，此操作在内核实现中只会在虚拟文件系统中将该 Socket 对应的引用计数减 1，并不会真正关闭该 Socket。这便需要用户态协议栈系统为 Socket 加入引用计数模式，以解决在多进程情况下 Socket 被错误关闭的问题。

真正实现用户态协议栈的高度兼容性还有更多的技术难题需要攻克，比如网络应用执行流异常崩溃时如何回收相应网络 fd 资源，如何支持 dup、dup2 这类对 fd 进行重定向的接口等。虽然实现高度兼容性困难重重，但其可以让高性能用户态协议的移植成本降到最低，造福网络应用开发者，这也是基础网络系统研究的意义所在。

5 结束语

本文从兼容性的各个方面对不同高性能网络协议栈进行对比分析。针对目前主流高性能协议栈兼容性不足的问题，本文提出在用户态实现高

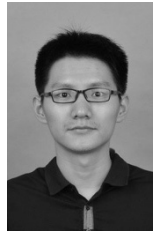
兼容协议栈的设想，为网络协议栈优化领域提供新的研究思路。

参考文献：

- [1] ZHANG H, ZHENG Z, XU S, et al. Poseidon: an efficient communication architecture for distributed deep learning on GPU clusters[C]//Annual Technical Conference, July 12-14, 2017, Santa Clara, CA, USA. [S.l.:s.n.], 2017: 181-193.
- [2] KERRISK M. The so_reuseport Socket option[EB]. 2013.
- [3] BARON J. Epoll: add EPOLLEXCLUSIVE flag[EB]. 2015.
- [4] Socket sendmsg MSG_ZEROCOPY[EB]. 2017.
- [5] PESTEREV A, STRAUSS J, ZELDOVICH N, et al. Improving network connection locality on multicore systems[C]//The 7th ACM European Conference on Computer Systems, Apr 10-13, 2012, Bem, Switzerland. New York: ACM Press, 2012: 337-350.
- [6] HAN S, MARSHALL S, CHUN B G, et al. MegaPipe: a new programming interface for scalable network I/O[C]//The 10th USENIX Conference on Operating Systems Design and Implementation, Oct 8-10, 2012, Hollywood, CA, USA. New York: ACM Press, 2012: 135-148.
- [7] LIN X, CHEN Y, LI X, et al. Scalable kernel TCP design and implementation for short-lived connections[J]. ACM SIGPLAN

- Notices, 2016, 51(4): 339-352.
- [8] THEKKATH C A, NGUYEN T D, MOY E, et al. Implementing network protocols at user level[J]. IEEE/ACM Transactions on Networking, 1993, 1(5): 554-565.
- [9] BASU A. A user-level network interface for parallel and distributed computing[C]//The 15th ACM Symposium on Operating Systems Principles (SOSP), Dec 3-6, 1995, Copper Mountain, Colorado, USA. New York: ACM Press, 1995: 40-53.
- [10] EDWARDS A, MUIR S. Experiences implementing a high performance TCP in user-space[J]. ACM SIGCOMM Computer Communication Review, 1995, 25(4):196-205.
- [11] MOGUL J, RASHID R, ACCETTA M. The packer filter: an efficient mechanism for user-level network code[C]//The Eleventh ACM Symposium on Operating Systems Principles, Nov 8-11, 1987, Austin, USA. New York: ACM Press, 1987: 39-51.
- [12] Intel. Data plane development kit project page[EB]. 2012.
- [13] RIZZO L. Netmap: a novel framework for fast packet I/O[C]//21st USENIX Security Symposium, Aug 8-10, 2012, Bellevue, WA, USA. New York: ACM Press, 2012: 101-112.
- [14] NTOP. Multi-10 Gbit RX/TX packet processing from hosts and virtual machines[EB]. 2016.
- [15] JEONG E Y, WOOD S, JANSHEED M, et al. mTCP: a highly scalable user-level TCP stack for multicore systems[C]//USENIX Conference on Networked Systems Design & Implementation, Apr 2-4, 2014, Seattle, WA, USA. New York: ACM Press, 2014: 489-502.
- [16] BELAY A, PREKAS G, KLIMOVIC A, et al. IX: a protected dataplane operating system for high throughput and low latency[C]//USENIX Conference on Operating Systems Design & Implementation, Oct 6-8, 2014, Broomfield, CO, USA. New York: ACM Press, 2014: 49-65.
- [17] PREKAS G, KOGIAS M, BUGION E. ZygOS: achieving low tail latency for microsecond-scale networked tasks[C]//The 26th Symposium on Operating Systems Principles, Oct 28-31, 2017, Shanghai, China. New York: ACM Press, 2017: 325-341.
- [18] PETER S, LI J, ZHANG I, et al. Arrakis: the operating system is the control plane[J]. ACM Transactions on Computer Systems (TOCS), 2016, 33(4): 11.
- [19] HONDA M, HUICI F, RAICIU C, et al. Rekindling network protocol innovation with user-level stacks[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(2): 52-58.
- [20] YASUKATA K, HONDA M, SANTRY D, et al. StackMap: low-latency networking with the OSstack and dedicated NICs[C]//2016 USENIX Annual Technical Conference, Jun 22-24, 2016, Denver, CO, USA. New York: ACM Press, 2016: 43-56.
- [21] POPE S, RIDDOCH D. Introduction to OpenOnload—building application transparency and protocol conformance into application acceleration middleware[R]. 2011.
- [22] PKELSEY. A library version of FreeBSD's TCP/IP stack plus extras[EB]. 2016.
- [23] TECENTE. F-stack[EB]. 2018.
- [24] SCYLLADB. High performance server-side application framework[EB]. 2014.
- [25] YULINTAO. Nginx with mTCP and DPDK[EB]. 2016.
- [26] SHALEV L, SATRAN J, BOROCIK E, et al. IsoStack-highly efficient network processing on dedicated cores[C]//USENIX Annual Technical Conference, June 23-25, 2010, Boston, MA, USA. New York: ACM Press, 2010: 5.

[作者简介]



姜惠友（1994- ），男，清华大学计算机系硕士生，主要研究方向为用户态网络协议栈。



李峻峰（1992- ），男，清华大学计算机系博士生，主要研究方向为网络编码和网络协议栈性能优化。



李丹（1981- ），男，清华大学计算机系长聘副教授、博士生导师。教育部“长江学者”（青年学者）奖励计划入选者，获得国家自然科学基金委员会“优秀青年科学基金”资助，国家“973”计划项目（青年科学家专题）首席科学家。主要研究方向为互联网体系结构。