计算机科学
COMPUTER SCIENCE

vsocket:一种基于RDMA的兼容标准套接字加速方法

陈云芳, 茆昊天, 张伟

引用本文

陈云芳, 茆昊天, 张伟. vsocket:一种基于RDMA的兼容标准套接字加速方法[J]. 计算机科学, 2023, 50(10): 239-247.

CHEN Yunfang, MAO Haotian, ZHANG Wei. vsocket:an RDMA-based Acceleration Method Compatible with Standard Socket [J]. Computer Science, 2023, 50(10): 239-247.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220800048

vsocket:一种基于 RDMA 的兼容标准套接字加速方法

陈云芳　茆昊天　张　伟
南京邮电大学计算机学院　南京 210000
(chenyf@njupt.edu.cn)

摘　要　为了兼容 Linux 标准套接字,同时利用 RDMA 提高使用套接字的程序的性能,提出在上层应用与底层 RDMA 之间搭建一个中间件——Viscore Socket adaptor(简称 vsocket);通过拦截 socket API,将上层应用通过套接字收发的数据流无缝转接到 RDMA 承载上。vsocket 绕过管理收发缓冲区的内核,针对 TCP 和 UDP 分别实现了用户空间的内存管理机制,使用 RC 类型的 RDMA 网络支持 TCP 加速,使用 UD 类型的 RDMA 网络支持 UDP 加速,并重用 Linux UDP 来辅助其路由。实验结果表明 vsocket 能够保证 Linux 标准套接字接口的兼容性,提升网络性能,摆脱 Linux 内核网络协议栈的限制,改善收发数据的延迟与带宽。

关键词:套接字;远程内存直接访问;协议加速
中图法分类号　TP393

vsocket:an RDMA-based Acceleration Method Compatible with Standard Socket

CHEN Yunfang,MAO Haotian and ZHANG Wei
School of Computer Science,Nanjing University of Posts and Telecommunications,Nanjing 210000,China

Abstract　In order to be compatible with Linux standard sockets and utilize RDMA to improve the performance of programs using sockets,this paper proposes to construct a middleware Viscore Socket adaptor,referred to as vsocket between the upper-layer application and the underlying RDMA. By intercepting the socket API,we seamlessly transfer the data stream sent and received by the upper-layer application through the Linux socket to the RDMA bearer. The vsocket bypasses kernel and implements memory management mechanism in user space for TCP and UDP. It utilizes RC type RDMA network to support TCP acceleration,uses UD type RDMA network to support UDP acceleration,and reuses Linux UDP to assist routing. Experimental results show that vsocket can ensure the compatibility of the Linux standard socket interface,get rid of the limitation of the Linux kernel network protocol stack,and improve the network performance.

Keywords　Socket,Remote-direct memory access,Protocol acceleration

1　引言

随着高性能计算、深度学习、大数据等应用的高速发展,用户对网络传输速率和延时的要求不断提升。网络传输速率从 25Gbps/50Gbps/100Gbps/200Gbps/400Gbps 一路飞速发展,并向着更快的 1.6Tbps 发展[1]。在网速越来越快而现代 CPU 频率已经趋于稳定的不平衡现状下,使用价格高昂的 CPU 来进行海量网络处理已经难以维系。现代 CPU 需要 10～15ns 来访问 L3 Cache,而 400Gbps 的网络仅需 1.2ns 便可传送 64Bytes 的消息[2]。此外,通过 CPU 访问内存、拷贝数据的开销在很多应用中占据了极大的比例[3],旁路 CPU 已经成为一种重要的优化手段。因此,能够满足高速网络处理需求、卸载不适合 CPU 的处理任务的 RDMA(Remote-Direct Memory Access)网络应运而生。我们预计,RDMA 将在高性能计算和智能计算等诸多场景中发挥更加重要的作用。

相比传统 DMA(Direct Memory Access)的内部总线 IO,RDMA 通过网络在两个端点的应用软件之间实现内存的直接传递;而相比传统的网络传输,RDMA 又无需操作系统和协议栈的介入。RDMA 具有三大特性:CPU 卸载、绕过内核、零拷贝。RDMA 可以轻易实现端点间的超低延时、超高吞吐量传输,且几乎不需要操作系统和 CPU 参与。在高速网络环境下,CPU 不必再为网络数据的处理和搬移耗费过多资源。

Linux 套接字(socket)是应用程序、容器和主机之间的标准通信原语,它利用传输层的"协议+端口"唯一标识主机中的应用程序(进程),网络中的进程通信就可以利用套接字这个标识与其他进程进行交互。通常,服务器首先创建一个套接字描述符 listenfd 用于监听端口和接收新连接,然后创建事件描述符 eventfd(使用 select,poll 或者 epoll)用于接收新连接事件和发送/接收数据事件,之后进入事件主循环,根据接收到的事件执行对应的动作。现代 Linux 套接字通常具备寻址、提供可靠服务或不可靠服务、支持事件驱动的 IO 多路复用模型等特性,即操作系统会通知应用程序发生 IO 事件的文件描述符。

RDMA 网络的消息服务建立在通信双方本地端和远端

到稿日期:2022-08-05　返修日期:2022-12-09
通信作者:张伟(zhangw@njupt.edu.cn)

应用之间创建的 channel-IO 连接之上。当需要通信时，就会创建一条 Channel 连接，每条 Channel 的首尾端点是两对 Queue Pairs(QP)，每对 QP 由 Send Queue(SQ)和 Receive Queue(RQ)构成，这些队列管理着各种类型的消息。QP 会被映射到应用的虚拟地址空间，使得应用可以直接通过它访问 RDMA 网卡。除了 QP 描述的两种基本队列之外，RDMA 还提供完成队列(Complete Queue，CQ)，CQ 用于通知用户 QP 上的消息已经被处理完。

在传统网络中，为了发送数据，主机应用程序需要将数据从用户空间拷贝到内核空间，同时要进行数据报文的封装和解析，然后才能被网卡访问，通过网卡发送到对端。对端主机则进行相反的过程，具体过程如图 1 所示 。RDMA 协议族完全绕过内核，能够节省一次内核空间与用户空间之间数据拷贝的开销(zero-copy)，并且通过硬件支持 RDMA 协议，将原本在软件实现的内核网络协议栈实现在专用硬件上，即支持 RDMA 的网卡，能够进一步降低 CPU 的负担，具体过程如图 2 所示。

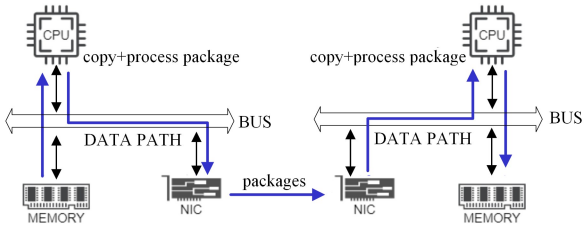

图 1 传统网络数据传输过程

Fig. 1 Data transfer process of traditional network

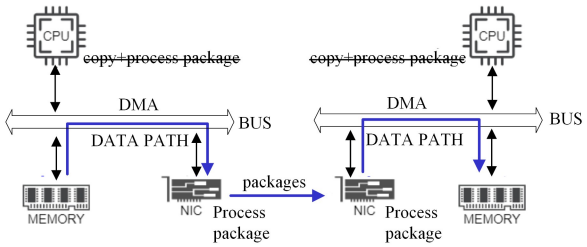

图 2 RDMA 网络数据传输过程

Fig. 2 Data transfer process of RDMA network

常用的 RDMA 网卡硬件根据其支持的 RDMA 协议不同分为 IB 卡和 RoCE 卡两类。IB 网卡需要安装专用的驱动软件，并与专用的 IB 交换机连接，部署成本较高。RoCE 网卡可以兼容传统以太网卡操作，支持 ip 和 ifconfig 等系统管理命令。上层应用可以同时访问以太网卡和 RoCE 网卡，通常使用通用套接字接口访问以太网卡，使用专用的 RDMA Verbs 接口访问 RoCE 网卡。当前大量应用基于 Linux 标准套接字接口开发，若想要直接使用 RDMA 协议，通常需要重写应用程序代码、更换物理设备，这需要投入巨大的人力和物力，开发周期长且稳定性难以保证。因此本文希望设计出一个能够结合 Linux 套接字系统和 RDMA 高速网络协议的高性能套接字系统，让应用程序的流量无缝透明地转接到 RDMA 载体上，使得应用程序不需要修改代码就能够享受 RDMA 高速网络的优势。

现代软件系统采用的技术栈大多是以太网卡支持的 TCP/IP 协议栈，应用层编程采用套接字系统。要将技术栈转为 RoCEv2 网卡支持的 RDMA 协议栈，需要对应用程序的底层通信进行修改，而 RDMA 的 Verbs API 使用复杂，并且要与用户内存管理相结合，掌握其开发技术的工程师在数量和熟练程度上存在不足，这无疑会极大地阻碍 RDMA 技术的普及与应用速度。通过把 RDMA 的 Verbs API 等封装为通信中间件，将其作为统一的公用的基础设施，面向不希望修改应用系统的用户，vsocket 可以有效降低用户开发难度，提高开发效率。为了保持 Linux 套接字系统的诸多特性，需要适配好 Linux 套接字编程规范和 RDMA 编程规范之间的差异，这种差异本质上是通信实体的差别。已有的许多研究工作展现了绕过内核网络协议栈的优越之处，本文的工作也完全位于用户空间，将 Linux 套接字系统的通信实体与 RDMA 规范的通信实体相关联，并提供有效的内存管理机制。当这些工作被完成，RDMA 网卡硬件及其协议就可以发挥其高性能优势，透明地加速使用 Linux 套接字系统的应用程序。

本文的工作让使用 Socket API 的应用在不修改代码的情况下也能使用 RDMA 协议，本文称之为 vsocket——一个处于 Socket API 和 Verbs API 之间的中间件。vsocket 的主要创新之处有：(1)vsocket 作为软件中间件，上层兼容 Linux 套接字，底层支持 RDMA 高性能网络协议栈，利用 RDMA 对套接字应用进行透明加速；(2)其完全绕过内核网络协议栈，将内核网络协议栈的内存管理工作搬移到用户空间。

2 基本概念

2.1 RDMA 协议

国际组织 IBTA(InfiniBand Trade Association)最早提出了 InfiniBand 协议，它是一整套完整的链路层到传输层的规范，但是其无法兼容现有以太网，除了需要支持 IB 的网卡之外，企业部署还需要重新购买配套的交换设备。

iWARP(internet Wide Area RDMA Protocol)协议也被称为 RDMA over TCP 协议，它基于 TCP 来承载 RDMA 协议，可以在标准以太网环境中使用 RDMA。由于 TCP 的性质，相比 IB 和 RoCE，iWARP 具有更好的可靠性，然而 TCP 的复杂性和高开销也导致了性能的损失。

RoCE 协议使用以太链路层来替代了 IB 链路层，被认为是 IB 的低成本解决方案，将 IB 的报文封装成以太网包进行收发。为了突破二层网络的限制，IBTA 进一步提出了 RoCE v2 协议，如图 3 所示，RoCE v2 使用 UDP＋IP 作为网络层，使得数据包可以在三层网络被路由。由于 RoCE v2 可以使用以太网的交换设备，因此在企业中应用较多。

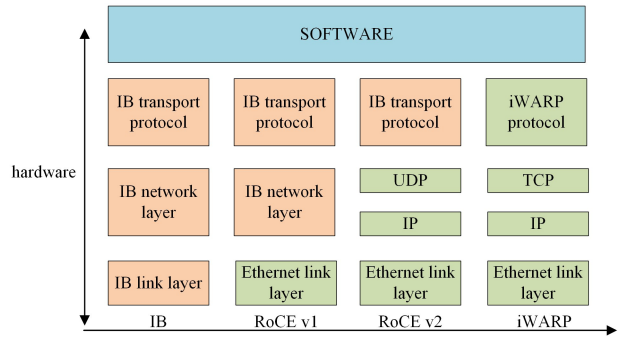

图 3 RDMA 主流协议

Fig. 3 Mainstream RDMA protocol

2.2 RDMA 通信原语

RDMA 支持 RC（Reliable Connection），UC（Unreliable Connection）和 UD（Unreliable Datagram）3 种通信模式。RC 模式保证报文传输的正确性，支持 ack 确认和重传等机制；UC 模式需要建链，报文不携带地址信息，不支持 ack 确认和重传机制，不保证报文传输的正确性；UD 模式不需要建链，报文携带寻址信息，不支持 ack 确认和重传机制，不保证对端能正确接收，并且每个报文不能大于 MTU（Maximum Transmission Unit）限制。RC，UC 和 UD 这 3 种模式的稳定性依次下降，执行效率依次升高。

RDMA 协议定义了两种通信原语：双边原语，如 send 和 recv，需要 CPU 参与传输过程；单边原语，如 write 和 read，能够直接访问远端内存而远端 CPU 无感知，单边原语是 RDMA 规范中最突出的特性。RC 模式支持全部原语，UC 模式不支持 read，UD 模式仅支持 send 和 recv 双边操作，vsocket 只利用 RC 和 UD 模式来加速 TCP 和 UDP 流量。

上层应用通过 wr 工作请求来给硬件下发任务。在 send 和 recv 操作中，不止发送端需要下发 wr，接收端也需要下发 wr 来通知硬件存放数据的内存位置。发送端并不知道发送的数据会存放在何处，每次发送数据，接收端都要提前准备好存放数据的内存，接收端 CPU 自然会感知这一过程。

RDMA 的 write 和 read 操作是本端内存主动写入和读取远端内存的行为，除了准备阶段，远端 CPU 不参与也不感知。但两端主机需要在准备阶段交换可用内存的地址和“钥匙”，相当于获得这块内存的读写权限，取得权限之后，两端主机就可以像访问本地内存一样直接读写对端的内存，这是 RDMA 内涵所在。用于 RDMA 的内存的地址对于上层应用来说是虚拟地址，因此可以很方便地进行操作，虚拟地址与物理地址的映射关系保存在 RDMA 网卡中，由硬件负责地址转换。“钥匙”的交换需要两端 CPU 的允许，一旦完成准备工作，RDMA 就可以发挥其优势：在高速读写大量数据的同时解放 CPU。

3 相关工作

在 RDMA 出现以前，大多数网络通信均基于 Linux 内核网络协议栈，其上层通常是由用户态 libc 提供的 SocketAPI，下层是由 Linux 内核提供的网络协议栈实现。而 RDMA 协议的接口是 VerbsAPI，这些 API 与 SocketAPI 互不兼容，如图 4 所示。

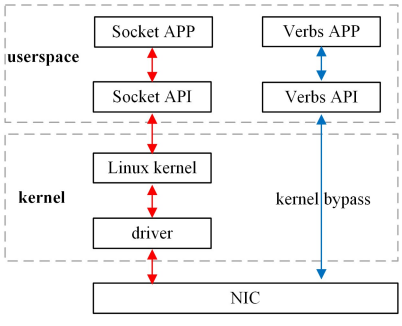

图 4 Linux socket API 与 Verbs API

Fig. 4 Linux socket API and Verbs API

基于 socket 编写的应用往往体量大、逻辑复杂，如果想要利用 RDMA 高速网络的优势，那么改写代码与调试的工作量将巨大且复杂。由于 RDMA 完全绕过内核，为了支持 TCP 和 UDP，在 RDMA 中对应 RC 和 UD 类型，需要在用户空间完成数据包的路由与内存管理工作。

学术界和工业界已经提出了若干高性能套接字系统。第一类工作聚焦优化内核 TCP / IP 协议栈，FastSocket[4] 修改优化了内核网络协议栈，使其具备高度可伸缩能力，能够更好地发挥多核 CPU 性能。MegaPipe[5] 将内核和用户空间之间用于交换异步 I/O 请求和完成通知的通道抽象化，并通过分区监听套接字、轻量级套接字和批处理系统调用实现性能提升。StackMap[6] 提出可以将高性能内核旁路的大多数技术集成到操作系统堆栈中，使得应用程序在获得高性能的同时能够享受现代 TCP/IP 内核协议栈的特性，但该方法无法提供良好的兼容性。第二类工作聚焦于将 TCP/IP 协议栈搬移到用户空间，Seastar[7]，F-stack[8] 和 LibVMA[9] 都提供了位于用户空间的高性能 TCP/IP 堆栈，前两者提出了新的 API，因此需要修改应用程序，LibVMA 符合标准套接字 API。用户空间 TCP/IP 协议栈提供了比 Linux 更好的性能，但它仍然无法和 RDMA 竞争。第三类工作是将传输层卸载到网卡硬件，TCP 卸载引擎[10] 将部分或全部的 TCP/IP 协议栈卸载到网卡上，但由于通用处理器的性能按摩尔定律迅速增长，这些专用硬件的性能优势有限。近年来，RDMA[11] 在数据中心得到广泛应用，它提供了单边操作和双边操作两种原语，双边操作类似于套接字的发送接收语义。为了使套接字应用程序能够使用 RDMA，Rsocket[12] 利用 API 劫持的方式将套接字操作转换为 RDMA 双边操作，然而 Rsocket 存在较多漏洞，甚至无法支持标准的 TCP 或 UDP 的性能测试程序。SocksDirect[13] 利用共享内存和 RDMA 分别加速主机内部和主机间通信，并且解决了 fork 和 epoll 等兼容性难题。它们或专注于优化内核 TCP/IP 协议栈，或专注于实现用户空间 TCP/IP 协议栈，或专注于将传输层下沉并针对 TCP 解决兼容性问题。以上工作通常都使用拦截套接字转接到高性能通信系统上的方案，然而这些方法都没有关注或实现 UDP 流量的加速，本文的工作同样利用套接字拦截技术，但创新性地利用 RDMA 的高性能优势分别实现了加速 TCP 和 UDP 流量，在细节上提供了必要的流控方案和内存管理方案，创新性地提出并实现了重用标准 UDP 类型套接字支持加速 UDP 流量的方案。

还有一些工作旨在将 RDMA 技术与云计算和虚拟化相结合，RDMA 技术提供了高带宽和低延迟的网络传输，虚拟机和容器技术提供了云计算用户隔离、共享物理资源以及灵活的管理。针对不同的云计算场景，RDMA 虚拟化的方法也不同。从虚拟化实现上来看，可以分为硬件辅助的虚拟化方法（如 SR-IOV[14-15]）、软件辅助虚拟化方法（如 vRDMA[16] 和 vSocket[17]）、混合虚拟化方法（如 HyV[18-19] 和 virtio-RDMA[20-21]）、针对容器的虚拟化方法（如 FreeFlow[22]）、纯软件模拟的方法（如 SoftRoCE[23]）。

在实际应用领域，阿里巴巴的 X-RDMA[24] 通信中间件在其大型集群托管云存储和数据库系统中被大量部署和

使用。X-RDMA 集成了当前 RDMA 生态系统中所不具备的功能，将开发人员从复杂和不完美的细节中解放出来。X-RDMA 简化了编程模型，扩展了 RDMA 协议以实现应用感知，并提出了每台机器有数千个连接的资源管理机制。自 2016 年以来，X-RDMA 已经部署在阿里云的多个大型集群中逾4 000 台服务器上。与 RDMA 相比，它可以节省至少 70% 的开发和维护时间，有效提高性能和减少网络抖动，特别是当服务器处于压力下。百度已经在生产系统中使用了通用和支持 RDMA 的 RPC(Remote Procedure Call)，在 BRPC 中实现 RDMA 支持，在支持开发人员零重构成本的同时实现最佳性能。许多生产级应用程序可以通过 BRPC 使用 RDMA，只需添加几行代码就能实现显著的性能改善。Accelio 是 Mellanox早期的 RDMA 网络开源库，由于其不能很好地支持数据中心网络和 GPU Direct 等新技术，已被弃用。Mellanox 被 Nvidia 收购后，Accelio 被 UCX(Unified Communication X)开源库取代。UCX 是一个通信框架，适用于现代高带宽和低延迟网络。它提供一组抽象通信原语，充分利用了底层硬件资源与卸载加速机制，统一通信框架的内部封装了 Verbs API，在上层提供一组抽象易用的通信原语。Rsocket 由 intel 开发实现并开源，包含在 rdma-core 开源项目中。Rsocket 支持 BSD 许可协议，需要二次开发并作为产品发布。vsocket 作为基于 Rsocket 二次开发的成果，是套接字转换为 RDMA 的实现完善。当前主流 RDMA 技术的优缺点对比如表 1 所列。

表 1 主流 RDMA 技术的优缺点

Table 1 Advantages and disadvantages of mainstream RDMA technology

名称	公司	投入	难度	接口	开源	备注
X-RDMA	阿里巴巴	大	中	—	×	
BRPC	百度	大	中	RPC	×	整体开源，RDMA 未开源
Accelio	Mellanox	中	中	—	√	已废弃
UCX	Nvidia	大	中	—	√	不兼容套接字
Rsocket	Intel	中	中	套接字	√	兼容性较差
vsocket	Viscore	小	小	套接字	—	

4 基于 RDMA 的加速套接字

4.1 中间件 vsocket 通信库

Linux 网络协议栈与 RDMA 协议栈的设计理念不同，导致 socket API 与 Verbs API 接口存在巨大差异。为了实现基于 RDMA 的套接字加速，本文设计了一种中间件 vsocket 来适配这种差异。例如 Linux socket 可以使用 sendto 方法发送数据，RDMA 则使用 ibv_post_send 方法发送数据。

ssize_t sendto(int socket, const void * buf, size_t length, intflags, const struct sockaddr * dest_addr, socklen_t dest_len);

int ibv_post_send(struct ibv_qp * qp, struct ibv_send_wr * wr, struct ibv_send_wr * * bad_wr).

Linux 通信的两端是两个套接字，而 RDMA 没有套接字的概念，RDMA 通信的两端是两个 QP(Queue Pair)，所以首先要抽象出一种兼容的包含 QP 的套接字 rsocket，并且在通信之前，rsocket 需要创建好 QP。ibv_post_send 的第二个参数 *wr*(*work request*)是一个复杂的结构体。对于 UD 类型的 QP 而言，它包含用于路由的关键参数 *ah*(*address handler*)，*qpn*(*qp number*)，*key*；对于 RC 类型的 QP 而言，它只需要包含 *remote_addr* 和 *key* 用于寻址。系统需要在发送数据前准备好这些参数。而 sendto 只需要指定参数 *dest_addr* 保存目的地址，内核会根据该地址完成路由。另外，RDMA 还需要手动注册一块内存并将注册的内存与 QP 绑定，然后这块内存才能以 RDMA 的方式发送数据。简言之，Linux 网络协议栈的大量工作被交给内核，而 RDMA 需要手动介入，特别是和内存相关的很多工作。

图 5 给出了通过劫持 socket API 来改变程序的执行流程，原本调用 socket API 的函数(*socket*，*send*，*recv* 等) 不会进入内核的网络协议栈，而是进入了作为中间件的 vsocket 通信库，最终转换为使用 RDMA 提供的 Verbs API 接口，这种操作被称为 API 劫持。vsocket 利用 Linux 环境变量 *LD_PRELOAD* 执行拦截 API 的操作。

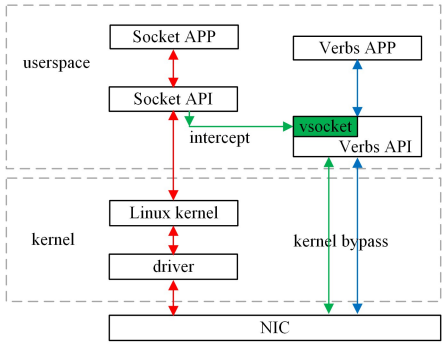

图 5 vsocket 设计架构

Fig. 5 Design architecture of vsocket

本文设计的 vsocket 是一个位于用户应用层和 Verbs 接口层之间的中间件，它可以在不修改应用的条件下实现将 TCP/UDP 流量转换为 RDMA 高速流量。vsocket 通过 API 劫持来实现不修改应用，把使用套接字应用的数据流转换为 RDMA 流量。由于其完全绕过了 Linux 内核网络协议栈，因此需要手动管理缓冲区内存。vsocket 分别针对 TCP 和 UDP 提供了不同的缓冲区管理机制。具体地，使用 RC 类型的 RDMA 网络支持 TCP 加速，拦截 TCP 连接转换为 RC 类型的 QP 连接；使用 UD 类型的 RDMA 网络支持 UDP 加速，并重用 LinuxUDP 来辅助其路由。

4.2 vsocket 通信库支持 TCP

4.2.1 拦截 TCP 连接

Linux 套接字的通信两端是两个 socket，而 RDMA 的通信两端是两个 QP，为了适配该差异并兼容 Linux 套接字，需要创建一种新的包含 QP 的“socket”。从应用程序的角度看，它依然是使用 socket 进行通信并且不需要做任何改变；从底层的角度看，它是 RDMA 高速网络的 QP 连接。具体过程如图 6 所示，通信两端在进行正式连接之前需要经过 vsocket 的初始化(rs_init)和绑定 RDMA 网卡地址(rdma_bind_addr)的过程，初始化过程为标准套接字分配好 RDMA 的软件资源，绑定过程为标准套接字指定可用的 RDMA 硬件资源，两端

套接字进入 rs_bound 状态。由于 RDMA 协议使用了另一套地址和路由方法，因此还需要通过 rdma_resolve_addr 操作将主机地址解析转换成 RDMA 可以识别的地址格式(Globally Unique Identifier，GID)，成功后套接字状态转换为 resolved_addr 地址已解析，通过 rdma_resolve_route 操作填入 RDMA 路由的“地址簿”，成功后套接字状态转换为 resolved_route 路由已解析。此时，两端套接字均处于准备好开始连接的状态。连接时，主机两端分别调用 RDMA 提供的连接方法(rdma_connect)进行连接，此时两端套接字分别处于 rs_accepting 和 rs_connecting 状态，由于我们已经成功给标准套接字分配了 RDMA 的资源，因此 RDMA 连接可以正常进行，最终连接建立完毕，两端套接字均处于 rs_connect_rdwr(已连接/可读/可写)状态，可以进行数据的读写，且对外暴露的仍然是标准套接字的接口，能够兼容已有的使用标准套接字接口的应用程序。

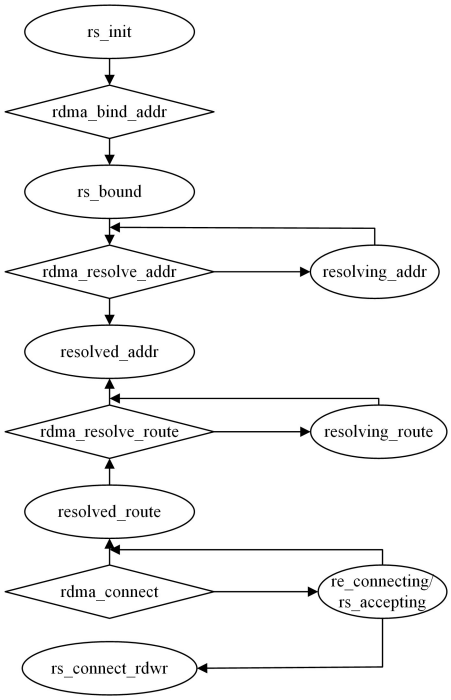

图 6 RDMA 连接建立过程

Fig. 6 RDMA connection establishment process

4.2.2 流量控制

vsocket 完全绕过了 Linux 内核网络协议栈，无法再依赖于内核 TCP 协议的流控机制。而 RDMA 协议栈被卸载到网卡，RDMA 的 RC 类型服务虽然具备了流控机制，但只能作用于网卡硬件，虽然保证了网卡上的内存不会被消耗殆尽，但主机接收缓冲区的内存却无人管理。另外，由于 RDMA 写会直接操作两端的内存，缺少相应的流控机制，因此接收缓冲区的数据还未及时交付给应用程序就可能被后续写入的数据覆盖。为了有效管理主机接收缓冲区的内存，vsocket 在用户空间实现了一套简单有效的流控机制，本文称之为 double check。让发送方在每次发送数据之前检查接收方的接收内存(就像检查本机内存一样)，只有在接收方内存可用的情况下发送方才开始发送数据。

如图 7 所示，当主机 A 与主机 B 建立起连接并开始发送数据，主机 A 作为发送方，需要提前知道数据发送到主机 B 上的内存的位置，而主机 B 作为接收方，则负责告知发送方存放数据的内存位置。(1)主机 B 的 remote_sgl 包含主机 A 上的 target_sgl 的 address，size 和 key 等信息，实际上这就是两块专用于收发流控信息的内存，RDMA 将这两块内存联系起来，主机 B 作为接收端负责向主机 A 更新自身的 rbuf(recv buffer)的可用情况；(2)主机 B check 自身可用 rbuf 的地址信息；(3)主机 B 根据 remote_sgl 取得主机 A 上的 target_sgl 的地址信息；(4)主机 B 根据取得的地址信息将自身可用 rbuf 的地址信息通过 RDMA 写操作写入主机 A 上的 target_sgl，此时，接收端的 rbuf 可用情况更新完毕，主机 A 上的 target_sgl 指向主机 B 的可用 rbuf；(5)主机 A 在发送数据前，check 本机 target_sgl 的内容，根据得到的地址信息发送数据到主机 B 上指定的 rbuf。由于 RDMA 写直接操作内存的特性，对于已连接的两端整体而言，主机 A check 本地 target_sgl 的动作实际上是第二次 check 了主机 B 上的可用 rbuf，因此被称为 double check。

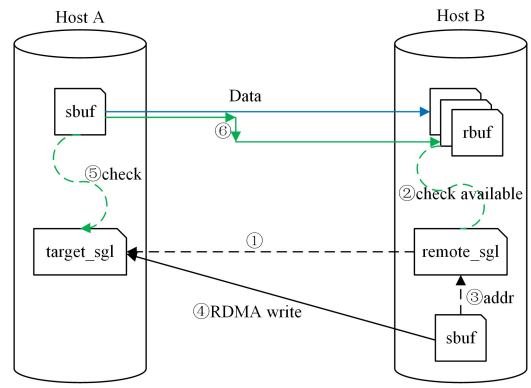

图 7 Double check 机制

Fig. 7 Double check mechanism

4.3 vsocket 通信库支持 UDP

4.3.1 辅助路由

Linux 协议栈里的 UDP 类型对应 RDMA 协议栈里的 UD 类型，两者十分相似，都是无连接且都不具备丢包重传机制，所以 vsocket 用 RDMA 协议栈中的 UD 取代 UDP。对比两者可以发现，UDP 使用 IP 进行主机间寻址，使用端口号进行主机内寻址，将 UDP 转化为 RDMA UD 需要改变其寻址方式。具体地，RDMA UD 需要使用 address handler(ah)和 qpn 分别进行主机间寻址和主机内寻址，为了保证正常的路由并兼容 UDP 操作(只有使用该套接字的进程才能访问收发的内存)，vsocket 重用 UDP 通信来交换 RDMA UD 需要的路由信息。双方路由信息完备后，RDMA 将发挥高性能优势，进行数据的收发，并且只有对应的使用套接字的进程才能访问这些数据。

重用 Linux UDP 辅助路由过程如图 8 所示，在阶段 I，主机 A 和主机 B 互不持有对方的 ah 和 qpn，无法为 UD 类型的 QP 提供路由服务。为此，在阶段 II，在拦截 socket 的同时，保留使用 Linux UDP 的套接字负责辅助路由，将需要发送的

数据和 ah 和 qpn 信息封装成 UDP 报文，并通过该套接字发送出去。在阶段Ⅲ，主机 A 和主机 B 都能获得路由所需的 ah 和 qpn。与此同时，接收方还需要处理阶段 II 接收到的 UDP 承载的数据。为了保证数据流量完全转移到 RDMA 的 QP 上，vsocket 将接收到的数据重新组包伪装成 RDMA 流量，转发至本机的发送队列 sq，并通过本机的接收队列 rq 进行接收。最后在阶段 IV，主机 A 和主机 B 不再需要 Linux UDP 的辅助，可以借助 ah 和 qpn 进行路由，直接使用 UD 类型的 QP 收发数据。

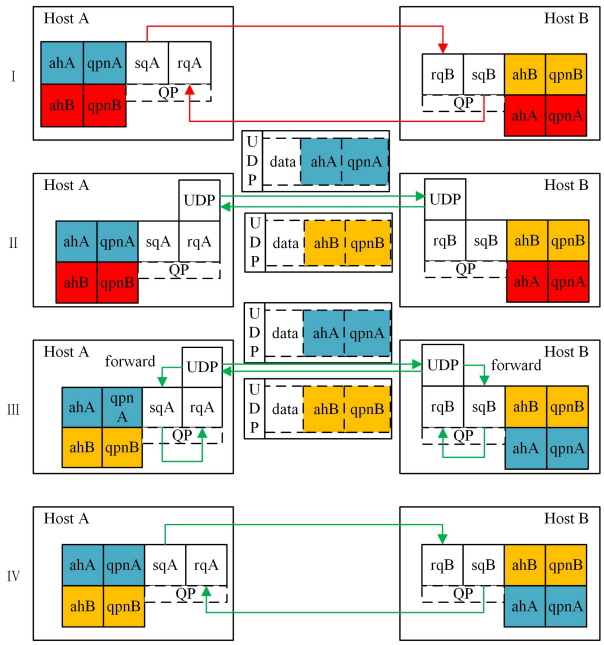

图 8 重用 Linux UDP 辅助路由

Fig. 8 Reuse Linux UDP to assist routing

4.3.2 UDP 收发缓冲区管理

由于 RDMA 完全绕过内核，内核不再帮助管理收发缓冲区，而 RDMA 协议只保证网卡内存的管理，因此 vsocket 需要为 UDP 转换为 RDMA 网络的 UD 类型的流量提供一套内存管理机制。在主机 A 进行 send 操作时，主机 B 必须提前显式下发 recv 请求来通知网卡硬件接收到的数据应存放到哪个地址。

vsocket 会提前下发并维护一定深度(接收队列的深度，为可控参数)的 recv 工作请求，每当发送或接收的工作请求被完成，都将更新对应的发送缓冲区或接收缓冲区的指针，使其始终指向下一个可用的字节或块。数据块有 3 种状态：used(已被预约，准备接收数据)；avail(未被预约)；dataavail(接收数据占用中)。如图 9 所示，在初始状态，主机 B 作为接收端提前 post 了 3 个 recv 工作请求，对应的 3 块接收内存置为 used，这样一旦接收到数据，网卡硬件就知道将接收到的数据的存放地址；之后，主机 A 下发一个 send 工作请求，自身对应的发送内存置为 used，当数据通过 RDMA 传输到主机 B 的 RDMA 网卡内存中，主机 A 得到工作完成的消息，将自身对应发送内存恢复成 avail，主机 B 上的一块标识为 used 的接收内存将被消耗置为 dataavail，等待网卡将数据 DMA 到这块内存，同时又下发一个新的 recv 请求；最终，数据到达指定的接收内存之后，交给上层的应用程序，该接收内存块重新恢复为 avail。

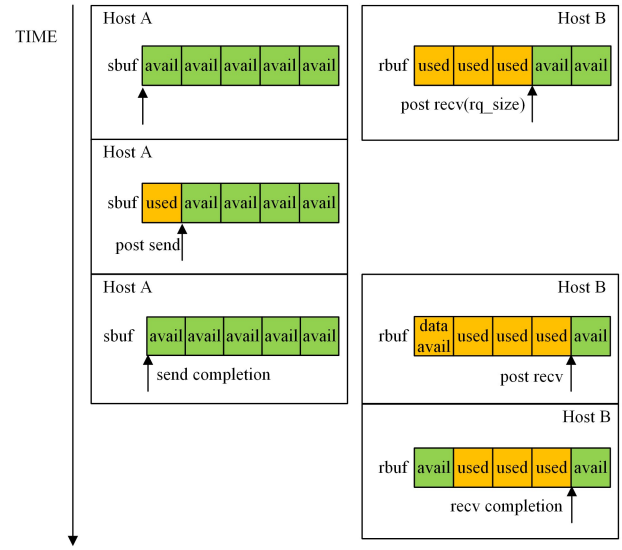

图 9 缓冲区管理

Fig. 9 Buffer management

但这种机制也有明显的缺陷，当 RDMA 高速网络加速数据传输过快，例如当主机 A 发送速率过快，主机 B 来不及接收而导致接收缓冲区充满时，将会出现明显的丢包现象。如图 10 所示，主机 A 已经发送了两块内存上的数据到主机 B 的网卡内存，主机 B 正忙于将网卡内存中的数据 DMA 发送到指定的接收内存块，这导致其状态为 data_avail，不可用于之后的 recv 请求，同时主机 A 下发了 3 个 send 请求，双方处于正在传输阶段；此时主机 B 已经没有接收内存可用，若主机 A 继续下发 send 请求，数据通过 RDMA 传输到达主机 B 网卡内存，然而由于没有可用的接收内存，数据将会在从网卡内存到主机接收缓冲区内存的过程中被丢弃。由于 UDP 本身是不可靠的传输，发送端尽可能地发送数据，接收端尽可能地接收数据，UDP 的丢包问题应当由使用 UDP 协议的上层应用来处理，vsocket 并不希望破坏这一语义。若希望得到可靠传输服务的应用，应当使用 TCP 协议或其他可靠的协议。

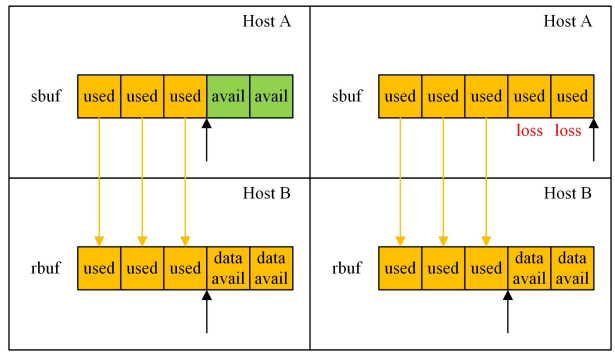

图 10 网络丢包

Fig. 10 Network packet loss

5 实验分析

为了避免复杂网络拓扑的路由产生的开销和损耗，实验使用两台配置了博通 25 GB 的 RDMA 网卡的台式机，两张

RDMA 网卡之间通过 DAC 线直连，分别运行使用 Linux 套接字系统和利用关键字 PRELOAD 加载 vsocket 通信库的实验程序。带宽与延迟是评价网络性能的两大主要指标，实验分别将 TCP、UDP、原生 RDMA 和 Nginx 作为 Web 服务器测试带宽和延迟。

实验使用 Iperf 网络性能测试程序进行 TCP 和 UDP 的带宽实验，Iperf 是美国伊利诺斯大学开发的一种开源的网络性能测试工具，可以用来测试网络节点间 TCP 或 UDP 连接的性能，包括带宽、抖动以及丢包率，其中抖动和丢包率适用于 UDP 测试，而带宽测试适用于 TCP 和 UDP。使用 Netperf 网络性能测试程序进行延迟实验，Netperf 也是一种测量网络性能的工具，主要针对基于 TCP 或 UDP 的传输。Netperf 根据应用的不同，可以进行不同模式的网络性能测试，实验利用其请求/应答(request/reponse)模式进行 TCP 延迟性能的测试。另外，实验还使用 perftest 进行原生 RDMA 性能的测试，perftest 编写了基于 Verbs API 的 RC/UD 等类型的性能测试工具，不存在 TCP/UDP 转换为 RC/UD 的性能损失，可以得到原生 RDMA 的性能指标。此类实验传输的流量大小和类型均由基准测试程序 Iperf 和 Netperf 生成，调整其数据包参数进行实验。

在实际场景应用方面，本文选择 Nginx 作为实验负载。通过部署 Nginx 作为静态资源服务器来访问服务器上的静态资源，在 Nginx 配置后可以通过 Nginx 来访问主机上的目标文件。实际上，我们同样可以将 Iperf 和 Netperf 当作使用 Linux 套接字系统的应用软件，这也体现了 vsocket 可以不加修改地透明加速并兼容应用程序所使用的标准套接字接口的特征。此类实验传输的数据是由 Linux 操作系统下的 truncate 指令生成的不同大小的文件。

5.1 TCP 加速实验

TCP 实验环境如表 2 所列。

表 2 TCP 实验环境

Table 2 TCP experimental environment

	server	client
OS	Centos 7.9.2009 kernel version:5.6.13	Centos 7.9.2009 kernel version:5.6.13
CPU	Intel(R) Core(TM) i3-10105 CPU @ 3.70GHz	Intel(R) Core(TM) i3-10100 CPU @ 3.60GHz
RNIC	Broadcom Inc. and subsidiaries BCM57414 NetXtreme-E 10Gb/25Gb RDMA Ethernet Controller	Broadcom Inc. and subsidiaries BCM57414 NetXtreme-E 10Gb/25Gb RDMA Ethernet Controller

带宽性能表现如表 3 所列，在数据包大小 2～4 000 Bytes 范围内，相比 Linux 套接字，vsocket 和原生 RDMA 都有显著的优势，在数据包大小为 4kB 时加速效果最为明显，并在 2kB 采样点最大化，vsocket 加速效果几乎是 Linux 套接字的 200%，同时也优于原生 RDMA 的带宽性能。原生 RDMA 的优越性能在更小包的情况下有更加明显的体现，说明 RDMA 技术在处理短小消息的场景下有着独特的优势。在数据包大小为 4kB～1MB 的范围内，3 种方法的测试带宽均已压满，在传输大包的场景下，3 种方法的带宽性能在 25 GB 网卡的环境下并没有明显的优劣之分。

表 3 TCP 带宽实验结果

Table 3 TCP bandwidth experimental results

	Bytes	Write_BW	Linux socket	vsocket
带宽/(Gbits/s)	2	0.10	0.02	0.03
	4	0.18	0.03	0.06
	8	0.36	0.07	0.12
	16	0.72	0.13	0.24
	32	1.44	0.27	0.48
	64	2.81	0.53	0.95
	128	5.28	1.03	1.90
	256	9.78	2.01	3.67
	512	15.25	3.92	7.10
	1 000	20.70	6.93	13.00
	2 000	22.08	12.30	22.80
	4 000	21.16	19.90	22.80
	8 000	22.70	23.50	22.90
	16 000	22.74	23.50	23.00
	32 000	22.82	23.50	23.00
	64 000	22.87	23.50	23.00
	128 000	22.88	23.50	23.00
	256 000	22.89	23.50	23.00
	512 000	22.89	23.50	23.00
	1 000 000	22.43	23.50	23.00

延迟性能表现如表 4 所列，与带宽性能类似，在数据包大小为 2kB～8MB 范围内，vsocket 可以将延迟降低到 Linux 套接字的 50%，但和原生 RDMA 依然存在较大差距，这也说明了 vsocket 将 TCP 流量拦截到 RDMA 流量的工作还存在着不小的损耗和开销，还有进一步优化的空间。在传输大包的场景下，vsocket 依然保持着低于 Linux 套接字的延迟，并且缩短了与原生 RDMA 的差距。

表 4 TCP 延迟实验结果

Table 4 TCP latency experiment results

	Bytes	Write_Lat	Linux socket	vsocket
延迟(usec)	2	4.32	70.64	36.37
	4	4.31	71.19	36.47
	8	4.34	69.94	36.31
	16	4.33	69.40	36.19
	32	4.36	70.59	36.24
	64	4.42	73.29	36.46
	128	5.39	71.61	37.84
	256	5.45	62.68	38.17
	512	5.68	73.25	38.75
	1 000	5.97	72.29	39.09
	2 000	6.19	75.36	39.65
	4 000	6.89	75.82	39.52
	8 000	8.24	76.82	40.31
	16 000	11.01	83.77	57.62
	32 000	16.45	92.89	48.34
	64 000	27.80	108.94	61.95
	128 000	50.33	126.72	89.09
	256 000	95.62	176.46	170.70
	512 000	186.28	267.86	233.02
	1 000 000	367.50	437.41	420.44

5.2 UDP 加速实验

UDP 实验环境如表 5 所列。由于 UDP 本身是不可靠的传输，发送端尽可能地发送数据，接收端尽可能地接收数据，vsocket 加速 UDP 导致接收端来不及接收从而出现丢包，因此接收端测试结果不具备参考价值。UDP 的丢包问题应当由使用 UDP 协议的上层应用来处理，我们并不希望破坏这一语义。对于希望得到可靠传输服务的应用，应当使用 TCP 协议或其他可靠的协议。实验只收集发送端的带宽数据。得益于 RDMA 的设计思想：完全在硬件中处理、解析数据包并

完全绕过内核，减少一次内核到用户空间的数据迁移，实现零拷贝。原生 RDMA 的 UD 类型网络的带宽性能优越，Linux 套接字性能较差，vsocket 加速方法能够提高 Linux 套接字的性能，但依然存在一定损耗，与原生 RDMA 性能相比依然存在差距，如图 11 所示。原生 RDMA 的 UD 类型网络的带宽是 LinuxUDP 类型的带宽的 500%～700%，是 vsocket 加速 UDP 带宽的 150%～200%，vsocket 利用 RDMA 的 UD 类型传输模式加速 Linux 套接字可以将 UDP 类型的发送带宽性能提升 400%。

表 5 UDP 实验环境

Table 5 UDP experimental environment

	server	client
OS	Centos 7.9.2009 kernel version:5.6.13	Centos 7.9.2009 kernel version:5.6.13
CPU	AMD Ryzen 9 3900X 12-Core Processor	Intel(R) Core(TM) i3-10105 CPU @ 3.70GHz
RNIC	Broadcom Inc. and subsidiaries-BCM57414 NetXtreme-E 10Gb/25Gb RDMA Ethernet Controller	Broadcom Inc. and subsidiaries BCM57414 NetXtreme-E 10Gb/25Gb RDMA Ethernet Controller

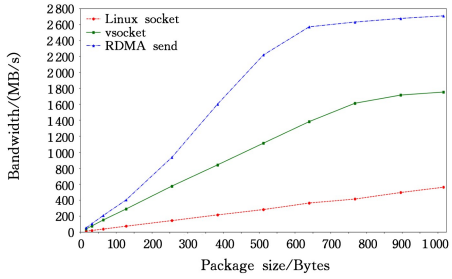

图 11 UDP 带宽实验结果

Fig. 11 UDP bandwidth experiment results

5.3 Nginx 加速实验

Nginx[25] 实验环境如表 6 所列。

表 6 Nginx 主要实验环境

Table 6 Nginx experimental environment

	server	client
OS	Centos 7.9.2009 kernel version:5.6.13	Centos 7.9.2009 kernel version:5.6.13
CPU	AMD Ryzen 9 3900X 12-Core Processor	Intel(R) Core(TM) i3-10105 CPU @ 3.70GHz
RNIC	Broadcom Inc. and subsidiaries BCM57414 NetXtreme-E 10Gb/25Gb RDMA EthernetController	Broadcom Inc. and subsidiaries BCM57414 NetXtreme-E 10Gb/25Gb RDMA Ethernet Controller

Nginx 是一款轻量级的 Web 服务器/反向代理服务器，它可以直接作为 HTTP server 部署，也可以作为反向代理服务器实现负载均衡。在 Linux 操作系统下，nginx 使用 epoll 或 poll 等事件模型，得益于此，nginx 在 Linux 操作系统下效率相当高。Nginx 作为 Web 服务器，处理事务的效率很高；作为代理服务器，可以实现反向代理加速。其高性能与稳定性并存，并且支持热部署，具备高可用性。

实验使用 Nginx 作为 Web 服务器对另一台主机提供 HTTP 服务，利用 wrk 工具测试 vsocket 加速 Nginx 前后的性能。wrk 是一款针对 HTTP 的基准测试工具，它能够在多核 CPU 的条件下，使用系统自带的 I/O 机制，如 epoll 和 poll 等，通过多线程和事件模式，对目标 Web 服务器产生大量的负载，并生成对应的测试报告。本文实验参考 nginx 官方网站的测试采样点[26]。实验在使用 poll 模型的 Nginx 上部署 vsocket，实现为应用程序透明加速的效果。

在较小目标文件实验条件下，在带宽方面，如图 12 所示，wrk 测试 Linux 套接字的 Nginx 带宽结果起伏较大且性能低下，甚至还会出现连接超时的情况，使用 vsocket 加速之后，Nginx 带宽性能远远优于使用 Linux 套接字的程序，连接更加稳定，并且在目标文件变大的情况下带宽趋于稳定。在实验环境下，vsocket 可以提高 150%～600%的带宽性能。

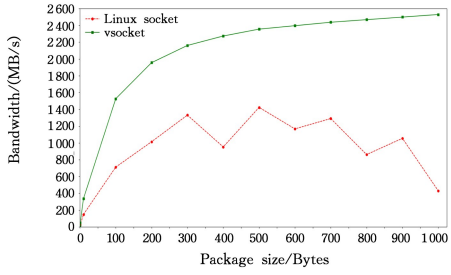

图 12 较小文件 Nginx 带宽实验结果

Fig. 12 Nginx bandwidth experiment results on small file

延迟方面，如图 13 所示，在所有实验采样点，vsocket 加速使用 Linux 套接字的 Nginx 均取得了显著效果，使用 vsocket 的 Nginx 的延迟最低只有使用 Linux 套接字的 Nginx 的 1/3。

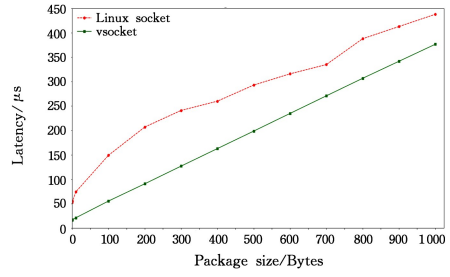

图 13 较小文件 Nginx 延迟实验结果

Fig. 13 Nginx latency experiment results on small file

在较大目标文件实验条件下，vsocket 加速效果不尽如人意。实验带宽性能如表 7 所列，使用 Linux 套接字和使用 vsocket 加速的 Nginx 均已压满带宽，趋于稳定，二者没有显著差别。

表 7 较大文件 Nginx 带宽实验结果

Table 7 Nginx bandwidth experiment results on large file

(单位:MB)

带宽/(GB/sec)	目标文件大小				
	10	100	300	500	1000
Linux socket	2.65	2.66	2.63	2.6	2.76
vsocket	2.65	2.67	2.66	2.66	2.68

实验延迟性能如表 8 所列，vsocket 加速效果聊胜于无。这样的现象基本符合原生 RDMA 测试的结果，即 RDMA 更加适合处理较小文件的网络传输。vsocket 作为基于 RDMA 技术的加速方法，无法超越 RDMA 的能力，在较小文件的实验中，vsocket 能够显著提高网络带宽，降低网络延迟，而在

较大文件的实验中，其性能基本与 Linux 套接字持平。

表 8 较大文件 Nginx 延迟实验结果

Table 8 Nginx latency experiment results on large file

（单位：MB）

延迟/ms	目标文件大小				
	10	100	300	500	1 000
Linux socket	3.68	36.75	111.50	187.50	354.35
vsocket	3.67	36.58	109.92	183.48	364.60

结束语 从兼容性和发挥高性能 RDMA 网络协议的角度出发，为了让使用 SocketAPI 的应用程序在不修改代码的情况下能无缝转接到 RDMA 承载上，摆脱 Linux 内核网络协议栈的限制，本文基于 Rsocket 开源项目开发了 vsocket 高性能通信库，针对 TCP/UDP 两种类型的网络流量进行加速，在保证兼容性和高性能的同时，分别实现了 API 劫持、TCP 连接转换、流量控制、UDP 辅助路由和缓冲区管理的设计，并在标准网络性能测试工具和实际应用场景中进行部署，取得了优越的加速效果。然而，由于 RDMA 的特性，vsocket 还无法完全兼容 fork 和 epoll 等特性，因此下一步研究的目标是在保证高性能的同时最大化兼容 Linux 套接字的特性。

参 考 文 献

[1] 2020 Ethernet Alliance Roadmap [EB/OL]. https://ethernetalliance.org/blog/2020/05/06/2020-ethernet-alliance-roadmap.

[2] HOEFLER T, GIROLAMO S D, TARANOV K, et al. sPIN: High-performance streaming Processing in the Network[C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2017: 1-16.

[3] MA X X, LU G, FU B Z, et al. Implementation method and performance analysis of non-contiguous data communication in network[J]. Chinese Journal of Computers, 2020, 43(6): 1123-1138.

[4] LIN X, YU C, LI X, et al. Scalable Kernel TCP Design and Implementation for Short-Lived Connections[J]. ACM SIGARCH Computer Architecture News, 2016, 44(2): 339-352.

[5] HAN S, MARSHALL S, CHUN B G, et al. MegaPipe: a new programming interface for scalable network I/O[C]// Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation. USENIX Association, 2012: 135-148.

[6] YASUKATA K, HONDA M, SANTRY D, et al. StackMap: Low-Latency Networking with the OS Stack and Dedicated NICs[C]// Usenix Technical Conference. 2016: 43-56.

[7] Seastar: High-performance server-side application framework [EB/OL]. http://seastar.io/.

[8] High-performance network framework based on dpdk[EB/OL]. http://f-stack.org/.

[9] MELLANOX. Messaging accelerator (vma)[EB/OL]. https://github.com/mellanox/libvma.

[10] CORPORATION M. Information about the tcp chimney offload [EB/OL]. https://support.microsoft.com/en-us/help/951037/information-about-the-tcp-chimney-offload-receive-side-scaling-and-net.

[11] InfiniBand Trade Association. InfiniBand Architecture specification, volume 1, release 1.0[EB/OL]. http://www.infinibandta.org

[12] Rsocket[EB/OL]. https://linux.die.net/man/7/.

[13] LI B, CUI T, WANG Z, et al. Socksdirect: datacenter sockets can be fast and compatible[C]// Proceedings of the ACM Special Interest Group on Data Communication. 2019.

[14] JOSE J, LI M, LU X, et al. SR-IOV Support for Virtualization on InfiniBand Clusters: Early Experience[C]// IEEE/ACM International Symposium on Cluster. ACM, 2013.

[15] MUSLEH M. Bridging the Virtualization Performance Gap for HPC Using SR-IOV for InfiniBand[C]// IEEE International Conference on Cloud Computing. IEEE, 2014.

[16] RANADIVE A, DAVDA B. Toward a Paravirtual vRDMA Device for VMware ESXi Guests[J]. VMware Technical Journal, Winter, 2012, 1(2): 2012.

[17] WANG D, FU B, LU G, et al. vSocket: virtual socket interface for RDMA in public clouds[C]// the 15th ACM SIGPLAN/SIGOPS International Conference. ACM, 2019.

[18] PFEFFERLE J, STUEDI P, TRIVEDI A, et al. A Hybrid I/O Virtualization Framework for RDMA-capable Network Interfaces[J]. ACM Sigplan Notices, 2015: 17-30.

[19] PFEFFERLE J. vVerbs, a paravirtual subsystem for RDMA-capable network interfaces[J/OL]. https://www.research-collection.ethz.ch/handle/20.500.11850/154568.

[20] FAN S, CHEN F, RAUCHFUSS H, et al. Towardsa Lightweight RDMAPara-Virtualization for HPC[C]// COSH/VisorHPC@ HiPEAC. 2017.

[21] MOUZAKITIS A, PINTO C, NIKOLAEV N, et al. Lightweight and Generic RDMA Engine Para-Virtualization for the KVM Hypervisor[C]// International Conference on High Performance Computing & Simulation. IEEE, 2017.

[22] KIM D, YU T, LIU H H, et al. Freeflow: software-based virtual RDMA networking for containerized clouds[C]// Networked Systems Design and Implementation. 2019.

[23] SoftRoCE: Software RDMA over Converged Ethernet[EB/OL]. https://github.com/SoftRoCE.

[24] MA T, MA T, SONG Z, et al. X-RDMA: Effective RDMA Middleware in Large-scale Production Environments[C]// IEEE International Conference on Cluster Computing(CLUSTER), 2019.

[25] REESE W. Nginx: The high-performance webserver and reverse proxy[J]. Linux Journal, 2008, 2008(173): 2.

[26] NGINX Plus Sizing Guide: How We Tested. [EB/OL]. https://www.nginx.com/blog/nginx-plus-sizing-guide-how-we-tested/#tests.

CHEN Yunfang, born in 1976, Ph. D, master supervisor. His main research interests include artificial intelligence algorithms, functional analysis of specific application areas, and application development using intelligent systems.

ZHANG Wei, born in 1973, Ph.D, Ph.D supervisor. His main research interests include intelligent perception and cognition under UAV platform, privacy protection and artificial intelligence security.

（责任编辑：何杨）