发表于《国际计算机通信杂志》,爱思唯尔,2006 年
DOI信息:10.1016/j.comcom.2006.11.001

Linux 网络的性能分析 – 数据包接收*

Wenji Wu+ , 马特·克劳福德、马克·鲍登
Fermilab, MS-368, PO Box 500, Batavia, IL, 60510 电子邮件:wenji@fnal.gov、crawdad@fnal.gov、bowden@fnal.gov 电话:+1 630-840-4541,传真:+1 630- 840-8208

抽象的

高能物理实验的计算模型正变得越来越全球分布和基于网格,这既是出于技术原因（例如,将计算和数据资源放置在彼此附近并满足需求）,也是出于战略原因（例如,为了利用设备投资）。为了支持这样的计算模型,网络和终端系统、计算和存储面临着前所未有的挑战。最大的挑战之一是在分布于世界各地的设施和计算中心之间可靠、高效地传输科学数据集（目前已达到数拍字节（1015字节)范围,预计在十年内将增长到艾字节）。网络和终端系统都应该能够提供支持高带宽、持续、端到端数据传输的能力。最近的技术趋势表明,尽管网络中使用的原始传输速度正在迅速增加,但微处理器技术的进步速度却已放缓。因此,与数据包传输所花费的时间相比,网络协议处理开销急剧增加,导致网络应用程序的吞吐量下降。越来越多的情况是,网络端系统而不是网络对网络应用程序的性能下降负责。本文从网卡到应用程序研究了Linux系统的数据包接收过程。我们开发了一个数学模型来表征 Linux 数据包接收过程。分析了影响Linux系统网络性能的关键因素。

关键词:Linux、TCP/IP、协议栈、进程调度、性能分析

一、简介

高能物理（HEP)实验的计算模型正变得越来越全球分布和基于网格,这既是出于技术原因（例如,将计算和数据资源放置在彼此附近并满足需求）,也是出于战略原因（例如,充分利用设备投资）。为了支持这样的计算模型,网络和终端系统、计算和存储面临着前所未有的挑战。

最大的挑战之一是在分布于世界各地的设施和计算中心之间可靠、高效地传输物理数据集（目前为数拍字节（1015字节)范围,预计在十年内将增长到艾字节）。网络和终端系统都应该能够提供支持高性能的能力

* 工作由美国能源部根据合同号 DE-AC02-76CH03000 支持。
+ 通讯作者

系统应该能够提供支持高带宽、持续、端到端数据传输的能力[1][2]。最近的技术趋势表明,尽管网络中使用的原始传输速度正在迅速增加,但微处理器技术的进步速度却已放缓[3][4]。因此,与数据包传输所花费的时间相比,网络协议处理开销急剧增加,导致网络应用程序的吞吐量下降。

越来越多的情况是,网络端系统而不是网络对网络应用程序的性能下降负责。

基于Linux的网络端系统已广泛部署在HEP社区（例如CERN、DESY、Fermilab、SLAC）。在费米实验室,数千个网络端系统正在运行Linux操作系统;其中包括计算群、触发处理群、服务器和桌面工作站。从网络性能的角度来看,Linux 代表了一个机会,因为它由于其开源支持以及 web100 和 net100 等支持网络工作堆栈参数调整的项目而易于优化和调整 [5][6]。本文从网卡到应用程序研究了Linux网络端系统的报文接收过程。我们使用成熟的技术而不是“尖端”硬件,以便专注于介于合理性能预期和实现之间的终端系统现象。我们的分析基于Linux内核2.6.12。第 1 层和第 2 层的网络技术采用以太网介质,因为它是最广泛和最具代表性的 LAN 技术。

此外,假设以太网设备驱动程序使用 Linux 的“新 API”或 NAPI [7][8],这会减少 CPU 上的中断负载。本文的贡献如下：(1)我们系统地研究了Linux内核中当前的数据包处理。 (2) 我们开发了一个数学模型来表征 Linux 数据包接收过程。分析了影响Linux系统网络性能的关键因素。

通过数学分析,我们将复杂的内核协议处理抽象并简化为三个阶段,围绕网卡驱动层的环形缓冲区和协议栈传输层的套接字接收缓冲区。 (3)我们的实验证实并补充了我们的数学分析。

本文的其余部分组织如下:第 2 节介绍了 Linux 数据包接收过程。第 3 节提出了一个数学模型来表征 Linux 数据包接收过程。分析了影响Linux系统网络性能的关键因素。在第 4 节中,我们展示了测试和补充我们的模型的实验结果,并进一步分析了数据包接收过程。第 5 节总结了我们的结论。

2. 收包流程

图 1 概括地展示了数据包从进入 Linux 终端系统到最终交付到应用程序的整个过程 [7][9][10]。一般来说,数据包的行程可分为三个阶段： · 数据包从网络接口卡 (NIC) 传输到环形缓冲区。 NIC 和设备驱动程序管理和控制此过程。

- 数据包从环形缓冲区传输到套接字接收缓冲区,由软件中断请求（软中断)驱动[9][11][12]。内核协议栈处理这个阶段。

- 数据包数据从套接字接收缓冲区复制到应用程序,我们将其称为数据接收过程。

在以下部分中,我们将详细介绍这三个阶段。

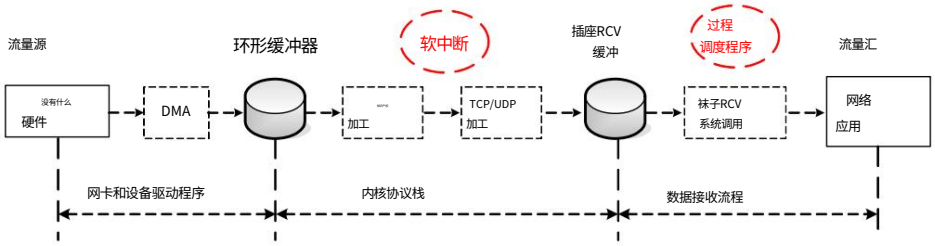

图1 Linux网络子系统:数据包接收流程

2.1 NIC 和设备驱动程序处理NIC 及其设备驱
动程序执行 OSI 7 层网络工作模型的第 1 层和第 2 层功能:从原始物理信号接收和转换数据包,并将其放入系统内存中,为更高层处理做好准备。 Linux 内核使用结构sk_buff [7][9] 来保存任何单个数据包,直至达到网络的 MTU（最大传输单元）。设备驱动程序维护这些数据包缓冲区的“环” ,称为“环缓冲区” ,用于数据包接收（以及用于传输的单独环）。环形缓冲区由设备和驱动程序相关数量的数据包描述符组成。为了能够接收数据包,数据包描述符应处于“就绪”状态,这意味着它已被初始化并使用空 sk_buff 进行预分配,该空 sk_buff 已内存映射到系统上 NIC 可访问的地址空间中/O 总线。当一个数据包到来时,接收环中的一个就绪数据包描述符将被使用,该数据包将通过 DMA [13] 传输到预先分配的 sk_buff 中,并且该描述符将被标记为已使用。使用过的数据包描述符应尽快重新初始化并用空的sk_buff重新填充,以供进一步传入的数据包使用。如果数据包到达并且接收环中没有就绪的数据包描述符,则该数据包将被丢弃。一旦数据包被传输到主内存中,在网络堆栈中的后续处理期间,数据包将保留在相同的内核内存位置。

图 2 显示了 NIC 和设备驱动程序级别的一般数据包接收过程。当接收到数据包时,它会被传输到主内存中,并且只有在内核可以访问该数据包后才会引发中断。当CPU响应中断时,驱动程序的中断处理程序被调用,其中

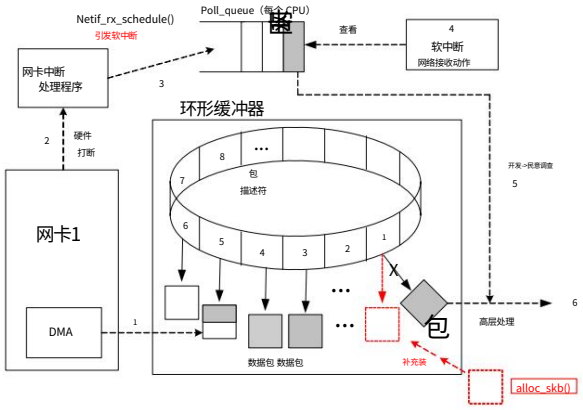

图2 网卡&设备驱动包接收

已安排软中断。它将对设备的引用放入中断 CPU 的轮询队列中。中断处理程序还会禁用 NIC 的接收中断，直到其环形缓冲区中的数据包被处理为止。

不久之后就会对软中断进行服务。 CPU通过调用设备驱动程序的poll方法轮询其轮询队列中的每个设备，以从环形缓冲区中获取接收到的数据包。每个接收到的数据包都会向上传递以进行进一步的协议处理。当接收到的数据包从其接收环形缓冲区中出队以进行进一步处理后，接收环形缓冲区中其相应的数据包描述符需要重新初始化并重新填充。

2.2 内核协议栈

2.2.1 IP 处理在处理从环形
缓冲区出列的每个 IP 数据包的软中断期间，将调用 IP 协议接收函数。该功能对 IP 数据包执行初始检查，主要涉及验证其完整性、应用防火墙规则以及处理数据包以转发或本地传递到更高级别的协议。对于每个传输层协议，都定义了相应的处理函数： tcp_v4_rcv()和udp_rcv()是两个示例。

2.2.2 TCP 处理当数据包向上交
给 TCP 处理时，函数tcp_v4_rcv()首先执行 TCP 头处理。然后确定与数据包关联的套接字，如果不存在则丢弃数据包。套接字有一个锁结构来保护它免受不同步的访问。

如果套接字被锁定，数据包将在积压队列中等待，然后再进行进一步处理。如果套接字未锁定，并且其数据接收进程正在休眠以获取数据，则数据包将添加到套接字的前置队列中，并将在进程上下文中而不是中断上下文中批量处理[11][12]。将第一个数据包放入预队列将唤醒休眠的数据接收进程。如果预队列机制不接受数据包，这意味着套接字未锁定并且没有进程正在等待其上的输入，则必须通过调用tcp_v4_do_rcv()立即处理数据包。

还调用相同的函数来排出积压队列和预队列。这些队列（预队列的情况除外）

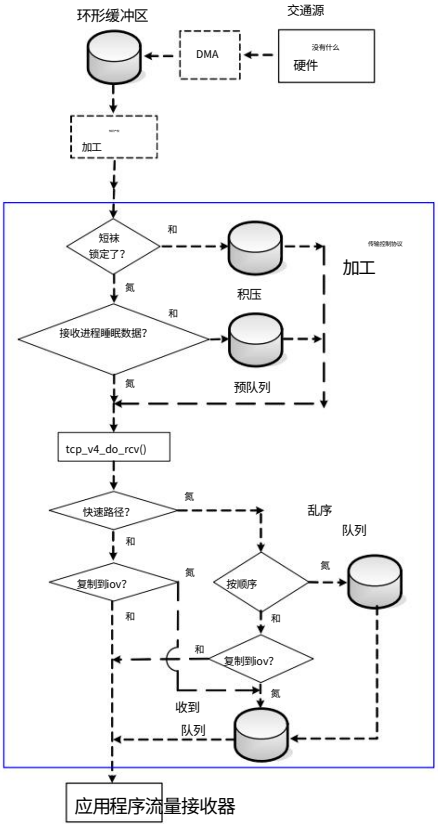

图 3 TCP 处理 - 中断上下文

溢出)在进程上下文中被耗尽,而不是软中断的中断上下文。在预队列溢出的情况下,这意味着预队列中的数据包达到/超过套接字的接收缓冲区配额,即使在中断上下文中,也应该尽快处理这些数据包。

tcp_v4_do_rcv()根据连接的 TCP 状态依次调用其他函数进行实际 TCP 处理。如果连接处于tcp_builted状态,则调用tcp_rcv_builted() ;否则,将执行tcp_rcv_state_process()或其他措施。 tcp_rcv_builted()执行关键的 TCP 操作:例如序列号检查、DupACK 发送、RTT 估计、ACK 和数据包处理。这里,我们重点关注数据包的处理。

在tcp_rcv_builted()中,当在快速路径上处理数据包时,将检查是否可以将其直接传递到用户空间,而不是添加到接收队列中。用户空间中的数据目的地由数据接收进程通过 sys_recvmsg 等系统调用提供给内核的iovec结构来指示。检查是否将数据包传送到用户空间的条件如下： · 套接字属于当前活动进程。 · 当前数据包是套接字序列中的下一个数据包； · 数据包将完全适合应用程序提供的内存位置;

当在慢速路径上处理数据包时,将检查数据是否按顺序排列（填充接收流中的空洞的开头）。与快速路径类似,如果可能的话,有序数据包将被复制到用户空间;否则,将其添加到接收队列中。失序数据包将添加到套接字的失序队列中,并安排适当的 TCP 响应。与积压队列、预队列和失序队列不同,接收队列中的数据包保证是有序的、已经被确认的并且不包含空洞。

当传入数据包填充数据流中前面的空洞时,失序队列中的数据包将被移动到接收队列。图3显示了中断上下文中的TCP处理流程图。

如前所述,积压和预队列通常在流程上下文中耗尽。
socket的数据接收过程

通过socket相关的receive系统调用从socket获取数据。对于 TCP,所有此类系统调用都会导致最终调用tcp_recvmsg(),这是 TCP 传输接收机制的顶端。如图所示

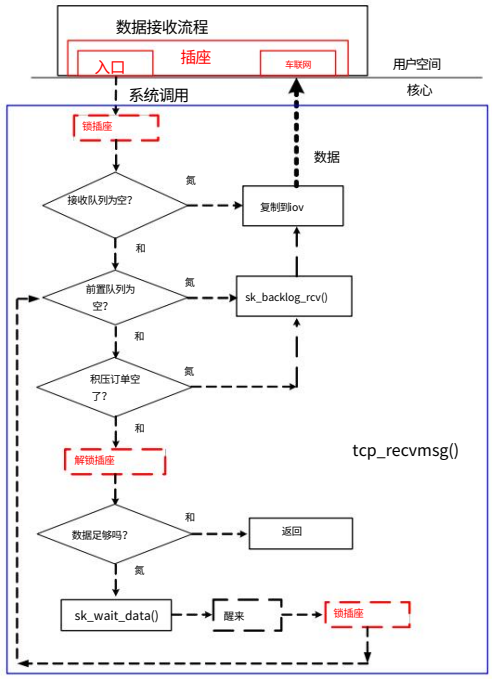

图 4 TCP 处理 – 进程上下文

在图 4 中,当调用tcp_recvmsg()时,它首先锁定套接字。然后它检查接收队列。由于接收队列中的数据包是保证有序、已确认且无漏洞的,因此接收队列中的数据会直接复制到用户空间。之后， tcp_recvmsg()将分别处理prequeue和backlog 队列（如果它们不为空）。两者都会导致调用tcp_v4_do_rcv()。之后,进行与中断上下文中类似的处理。 tcp_recvmsg()在返回给用户代码之前可能需要获取一定量的数据;如果所需的数据量不存在,则将调用sk_wait_data()使数据接收进程进入睡眠状态,等待新数据的到来。数据量由数据接收进程设置。在tcp_recvmsg()返回用户空间或数据接收进程进入睡眠状态之前,套接字上的锁将被释放。如图4所示,当数据接收进程从睡眠状态醒来时,需要再次重新锁定套接字。

2.2.3 UDP 处理当UDP 数据包从IP
层到达时,它被传递到udp_rcv()。 udp_rcv()的
任务是验证 UDP 数据包的完整性,并将一份或多份副本排队以传送到多播和广播套接字,以及将一份副本排队到单播套接字。
当将接收到的数据包放入匹配套接字的接收队列中排队时,如果该套接字的接收缓冲区配额空间不足,则该数据包可能会被丢弃。
套接字接收缓冲区内的数据已准备好传送到用户空间。

2.3 数据接收过程数据接收过程通
过套接字相关的接收系统调用,最终将数据包数据从套接字的接收缓冲区复制到用户空间。接收进程提供内存地址和要传输的字节数,或者在iovec 结构中,或者作为由内核收集到这样的结构中的两个参数。如上所述,所有与 TCP 套接字相关的接收系统调用都会导致最终调用tcp_recvmsg(),该函数将通过iovec从套接字的缓冲区（接收队列、预队列、积压队列)复制数据包数据。对于UDP,所有与套接字相关的接收系统调用都会导致最终调用udp_recvmsg()。当调用udp_recvmsg()时,接收队列中的数据通过iovec直接复制到用户空间。

3. 性能分析

基于第2节描述的数据包接收过程，数据包接收过程可以用图5的模型来描述。在数学模型中,网卡和设备驱动程序接收过程可以用令牌桶算法来表示[14],如果环形缓冲区中有可用的就绪数据包描述符,则接受数据包;如果没有,则丢弃它。其余的数据包接收过程被建模为排队过程[15]。

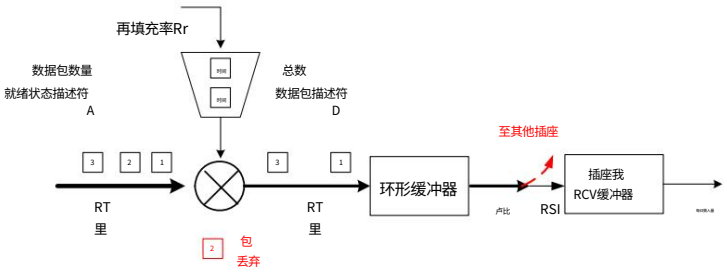

图5 数据包接收过程　数学模型

我们假设有多个传入数据流到达并定义以下符号： · tR)(, tR
时间 时间)(:提供和接受的总数据包速率（数据包/时间单位）； · tR)(,
Ri (t):数据流i的提供和接受的数据包速率（数据包/时间单位）；
· tR)(:时间t时已用数据包描述符的重新填充率（数据包/时间单位）； · D：
接收环形缓冲区中的数据包描述符总数；
· A : t)(t 时刻处于就绪状态的数据包描述符数量；
- t 分钟 :数据包进入系统和首次由软中断服务之间的最小时间间隔；
- ：NIC的最大数据包接收速率（Packets/Time Unit）；

· tR)(:内核协议数据包服务速率（数据包/时间单位）； · tR)(:流i的软中断数据包服务速率（数据包/时间单位）； · tR)(:流i的数据接收过程数据包服务速率（数据包/时间单位）； · tB)(:套接字i 在时间 t 的接收缓冲区大小（字节）；

- 量子比：Socket i 的接收缓冲区配额（Bytes）；

· N： 数据包接收期间可运行的进程数；
- P1 , … , PN :数据包接收期间N个可运行的进程；
- :数据流i的数据接收过程；

令牌桶算法非常适合 NIC 和设备驱动程序接收过程。在我们的模型中，接收环形缓冲区被表示为深度为D个令牌的令牌桶。处于就绪状态的每个数据包描述符都是一个令牌，授予接受一个传入数据包的能力。仅当重新初始化并重新填充使用的数据包描述符时，才会重新生成令牌。如果桶中没有令牌，传入的数据包将被丢弃。

那么，它有：

$$t > \forall 0 , \quad RT(t) = \begin{cases} RT(t), A(t) > 0 \\ 0, A(t) = 0 \end{cases} \tag{1}$$

因此，要允许数据包进入系统而不被丢弃，系统应满足条件： t >∀ 0

$$, \quad t)(> 0 \tag{2}$$

另外，还可推导出：

$$A(t) = D - RT(t)dt + Rf(t)dt , \forall t > 0 \tag{3}$$

从(1)和(3)可以看出，为了避免或最小化丢包，
NIC，系统需要提高其tR)(r和/或有效降低 D ，
)(时间 。

由于已使用的数据包描述符可以在其相应数据包从接收环形缓冲区出队以进行进一步处理后重新初始化和重新填充,因此tR)(r

取决于以下两个因素： (1) 协议包服务速率（tR ） s 。为了提高协议内核数据包服务率,已经提出了优化或卸载系统协议内核中的内核数据包处理的方法。例如,TCP/IP Offloading技术[16][17][18][19][20]旨在通过将任务转移到NIC或存储设备来释放CPU的一些数据包处理能力。 (2)系统内存分配状态。当系统面临内存压力时,新数据包缓冲区的分配很容易失败。在这种情况下,已使用的数据包描述符无法重新填充;的比率实际上是降低了。当所有处于就绪状态的数据包描述符都用完时， tR)(r

NIC 将丢弃更多传入数据包。 4.1 节中的实验将证实这一点。在不存在内存短缺的情况下,可以假设tRtR)()(s = r 。

D是NIC和驱动程序的设计参数。较大意味着 NIC 的成本增加。对于 NAPI 驱动程序,应足够大,以便在 NIC 接收环缓冲区中接收到的数据包出队并且重新初始化和重新填充接收环缓冲区中相应的数据包描述符之前,可以容纳更多传入数据包。在这种情况下， D

至少应满足以下条件以避免不必要的丢包：（4）

*研发t 分钟 最大限度

在这 t 分钟 是数据包进入系统之间的最小时间间隔,包括以下组成部分里,它首先由软中断提供服务。一般来说,[12][21]： · NIC 中断 t 分钟 调度时间

（NIDT）:当 NIC 中断发生时,系统必须在调用 NIC 数据包接收中断服务例程来处理它之前保存所有寄存器和其他系统执行上下文。 · NIC 中断服务时间(NIST):NIC 中断服务例程从NIC 检索信息并调度数据包接收软中断所用的时间。

其中,在给定硬件配置的情况下,NIDT 的值几乎恒定。但是,NIST 取决于 NIC 中断处理程序的长度。设计不当的 NIC 设备驱动程序可能会施加很长的 NIST 值并导致不可接受的大.对于给定的设计不良的 NIC 设备驱动程序,甚至可能导致接收环形缓冲区中的数据包丢失。 t 分钟 D ,

tR)(是提供的总数据包速率。通常,人们会尝试增加tR)(以最大化传入吞吐量。为了避免或最小化 NIC 造成的数据包丢失,减少tR)(似乎是一种不可接受的方法.不过,使用jumbo

框架 * [22][23][24]有助于保持传入字节速率,同时减少tR ）（避免 NIC 处的数据包丢失。使用 1Gb/s 的巨 到型帧可将最大数据包速率从每秒超过 80,000 个降低到每秒 14,000 个以下。由于巨型帧

* IEEE 802.3 以太网规定最大传输单元 (MTU) 为 1500 字节。但许多千兆位以太网供应商已遵循 Alteon Networks 的提议,支持超过 9000 字节的巨型帧大小。

减少Rmax ，从(4)可以看出,对D的要求可能会降低
巨型帧。

其余的数据包接收过程被建模为排队过程。在模型中,套接字i的接收缓冲区是一个大小为QBi的队列
。数据包被放入队列中
通过软中断,速率为(tR) 和 ，并被数据接收移出队列
处理速率为(tR) 的 。

对于流i,根据数据包接收过程,有： tRtR和tRtR)()(
)()(≤ 和 ≤ (5)

相似,可推导出：
乙(t) = Rs(t)d t t —tRdi( )d t (6)

在传输层协议操作中tB)(i起着关键作用。对于 UDP,当)(≥QBtB _ ，
套接字i的所有传入数据包都将被丢弃。在这种情况下,对丢弃的数据包进行的所有协议处理工作都将被浪费。无论是从网络端系统还是网络应用的角度来看,这都是我们极力避免的情况。

当接收缓冲区已满时,TCP 不会像 UDP 那样在套接字级别丢弃数据包。相反,它向发送方通告(tBQB)
－以执行流量控制。然而,当 TCP 套接字的接收缓冲区接近满时,向发送方通告的小窗口 (tBQB) 将限制
发送方的数据发送速率,从而导致 TCP传输性能下降[二、
25]。

从 UDP 和 TCP 的角度来看,提高(tBQB)的值是可取的,即： 二、 ，

QBi —tRsi( )d t + Rdi( )d t t (7)

显然,减少(tR) 来实现目标是不可取的。但这个目标可以通过提高QBi和/或(tR)来实现
的 。对于大多数操作系统， QBi 是可配置的。
对于 Linux 2.6， QBi 可通过 /proc/net/ipv4/tcp_rmem 进行配置,它是一个包含三个元素的数组,给出接收缓冲区大小的最小值、默认值和最大值。

为了最大化 TCP 吞吐量,配置 TCP 的经验法则是将其设置为端到端路径的带宽*延迟乘积 (BDP)（TCP
发送套接字缓冲区大小以相同方式设置）。这里的Bandwidth是指端到端路径的可用带宽； Delay为往返时间。根据上述规则,对于长、高带宽的连接,将设置得高。 IA-32 Linux系统通常采用QBi

3G/1G虚拟地址布局,3GB用户空间虚拟内存,1GB内核空间

空间[12][26][27]。由于这种虚拟地址分区方案,内核最多可以直接将896MB物理内存映射到其内核空间。这部分内存称为Lowmem。内核代码及其数据结构必须驻留在 Lowmem 中,并且它们不可交换。然而,为QBi分配的内存

s （和发送套接字缓冲区）

也必须驻留在 Lowmem 中,并且也是不可交换的。在这种情况下,如果设置为高（例如,5MB或10MB)并且系统有大量TCP连接QBi

系统（例如,数百个）,它很快就会耗尽 Lowmem。 “当你离开 Lowmem 时,就会发生不好的事情”[12][26][27]。例如,一个直接后果是导致网卡丢包:由于Lowmem内存不足,无法重新填充已使用的包描述符;当所有处于就绪状态的数据包描述符都用完时,更多传入数据包将在 NIC 处被丢弃。为了防止 TCP 过度使用 Lowmem 中的系统内存,Linux TCP 实现有一个控制变量 - sysctl_tcp_mem 来限制整个系统的 TCP 使用的内存总量。 Sysctl_tcp_mem可通过 /proc/net/ipv4/tcp_mem 进行配置,它是一个由三个元素组成的数组,给出了所有 TCP 套接字允许排队的最小值、内存压力点和最大页数。对于 IA-32 Linux 系统,如果设置为高,则sysctl_tcp_mem

量子比

应进行相应配置,以防止系统耗尽 Lowmem。
对于IA-64 Linux系统Lowmem没有这样的限制,所有安装的物理内存都属于Lowmem。但是配置QBi并且sysctl_tcp_mem仍然受到物理内存限制。

$tR_{的}$)(取决于数据接收进程本身和所提供的系统负载。所提供的系统负载包括所提供的中断负载和所提供的进程负载。这里,所提供的中断负载是指所有与中断相关的处理和处理（例如,NIC中断处理程序处理、数据包接收软中断处理）。在 Linux 等中断驱动的操作系统中,由于中断比进程具有更高的优先级,因此当提供的中断负载超过某个阈值时,用户级进程可能会缺乏 CPU 周期,从而导致减少(tR)

的 。在极端情况下,当用户级进程完全被中断抢占时， (tR) 将降至零。例如,在接收livelock [的

8],当非流量控制数据包到达太快时,系统将花费所有时间处理接收器中断。因此,它将没有剩余的 CPU 资源来支持将到达的数据包传送到数据接收进程,并且(tR) 降至零。对于网络负载较重的情况,通常采用以下方法来减少所提供的中断负载并节省网络应用程序的 CPU 周期： (1) 中断合并 (NAPI) [的

8],通过为每个中断处理多个数据包来降低数据包接收中断的成本。 (2) 巨型帧[22][23][24]。如上所述,巨型帧可以有效降低传入数据包速率,从而降低中断率和中断负载。具体来说,巨型帧将减少每字节产生的网络堆栈处理（软中断处理)开销。可以获得 CPU 利用率的显着降低。 (3) TCP/IP 卸载[17][18]。

在以下各节中,我们从提供的进程负载的角度讨论$_{的}$(tR),假设系统未过载。

Linux 2.6 是一个抢占式多处理操作系统。进程（任务）被安排以优先级循环的方式运行[11][12][28]。如图6所示，整个进程的调度是基于一个叫做runqueue的数据结构。本质上，运行队列跟踪分配给特定 CPU 的所有可运行任务。因此，系统中的每个 CPU 都会创建并维护一个运行队列。

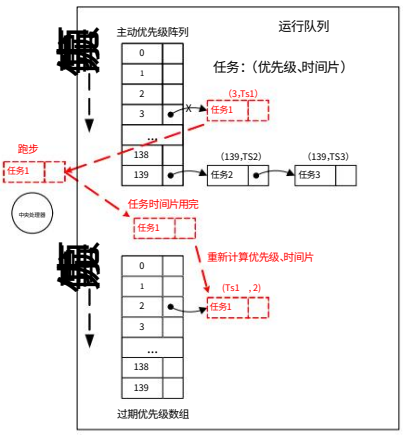

图6 Linux进程调度

每个运行队列包含两个优先级数组：活动优先级数组和过期优先级数组。每个优先级数组包含每个优先级的一个可运行进程队列。优先级较高的进程被安排首先运行。在给定的优先级内，进程被循环调度。 CPU 上的所有任务都从活动优先级数组开始。每个进程的时间片根据其nice值计算；当活动优先级数组中的进程用完其时间片时，该进程被视为过期。过期的进程将移至过期优先级数组+ 。在移动过程中，会计算新的时间片和优先级。

当活动优先级数组中不再有可运行的任务时，只需将其与过期优先级数组进行交换。

假设在数据接收期间，系统进程负载稳定。

有N 个正在运行的进程： ，；是数据P1的数据接收过程，该过程运行时PN的分组业务速率是恒定的。每个工

Pi 我

不会休眠（例如，等待 I/O），或者与它的时间片相比，休眠进程很快就会醒来。这样，数据接收进程P的运行模型可以如图7所示进行建模。进程运行时可能会发生中断。由于中断处理时间不按进程计费，因此我们这里不考虑中断。

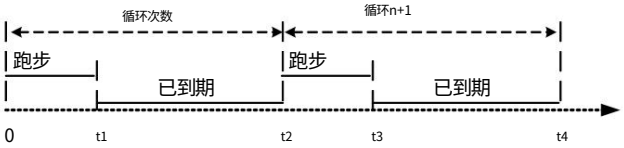

图7 数据接收进程运行模型

进一步，可推导出 的运行周期：

$$\sum_{j=1}^{N} P_{时间片j} \quad (8)$$

模型中，进程的运行周期为Timeslice)(Pi，且过期时间为：

$$\sum_{j=1}^{N} P_{时间片_j} - P_{timeslice} \quad (9)$$

+ 为了提高系统响应能力，交互式进程被移回活动优先级数组。

从进程的角度来看,进程的相对 CPU 份额为：

$$\text{时间片} \quad \left(\sum_{j=1}^{\text{氮}} / \right) \text{P时间片}\left(\quad j \right) \tag{10}$$

从(8)、(9)、(10)可以看出,当进程的nice值相对固定时(例如,在Linux中,默认的nice值为0),will的数量决定了进程的运行频率。圆周率 氮
昆西。

对于图 7 中的周期n ,我们有： 0 < t < t1

$$\text{Rdi}(t) = \begin{cases} \text{我}, \\ 0, & t1 < t < t2 \end{cases} \tag{11}$$

因此,要提高(tR)的速率,就需要增加数据接收进程的CPU占有率:要么增加数据接收进程的时间片/nice值,要么通过降低数据接收进程的运行频率来降低系统负载。实验N

4.3 节的结果将证实这一点。

提高tR速率的另一种方法是从程序员的角度提高数据包服务速率,可以采取以下优化来最大化 (1) 缓冲区 我。从对齐 [29][30]; (2) 异步 I/O [30]。 我:

4 结果与分析

我们在费米子网络上运行数据传输实验。在实验中,我们运行 iperf [31] 在两个计算机系统之间单向发送数据。 iperf

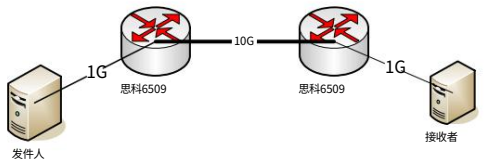

图8 实验网络及拓扑

在接收端是数据接收过程。如图8所示,发送方和接收方分别连接到两台Cisco 6509交换机。相应连接的带宽如标记的那样。发送方和接收方的特征如表1所示。

	发送器两	接收者 †
中央处理器	个 Intel Xeon CPU (3.0 GHz)	1 个 Intel Pentium II CPU (350 MHz)
系统内存	3829 MB	256MB
没有什么	Tigon、66MHz 的 64 位 PCI 总线插槽、1Gbps/秒、双绞线 Linux 2.6.12（3G/	Syskonnect、33MHz、1Gbps/秒的 32 位 PCI 总线插槽、双绞线 Linux 2.6.12（3G/
你	1G 虚拟地址布局）	1G 虚拟地址布局）

表 1 发送器和接收器特性

†我们在不同版本的 Linux 接收器上进行了实验,并获得了类似的结果。

为了研究详细的数据包接收过程,我们在Linux数据包接收路径中添加了仪器。此外,为了研究系统在各种系统负载下的接收性能,我们通过运行make –nj [11]将 Linux 内核编译为后台系统负载。不同的n值对应不同级别的后台系统负载,例如make –4j。为简单起见,它们被称为“BLn”。

后台系统负载意味着CPU 和系统内存上的负载。

我们运行ping来获取发送方和接收方之间的往返时间。最大 RTT 约为 0.5ms。端到端路径的BDP约为625KB。当TCP套接字的接收缓冲区大小配置得大于BDP时,TCP性能将不受TCP流控制机制的限制（小TCP套接字的接收缓冲区大小会限制端到端性能,读者可以参考[25][32]）。为了验证这一点,我们对等于或大于1MB的各种接收方缓冲区大小进行实验:发送方向接收方传输一个TCP流100秒,所有进程都以nice值0运行。实验结果如表2所示可以看出:当TCP套接字的接收缓冲区大小大于BDP时,获得了类似的结果（端到端吞吐量）。

	实验结果:端到端吞吐量 1M 10M 20M 40M 80M 310 Mbps				
接收缓冲区大小	309 Mbps 310 Mbps 309 Mbps 308 Mbps 64.7 Mbps 63.7 Mbps 63.9 Mbps 65.2				
BL0	Mbps 65.5 Mbps 30.7 Mbps 31 Mbps 31.4 Mbps 31.9 Mbps 3 1.9 Mbps				
BL4					
BL10					

表 2 不同套接字接收缓冲区大小的 TCP 吞吐量

以下实验中,除非特别说明,所有进程均以nice值为0运行； iperf的接收缓冲区设置为40MB。从系统级别来看， sysctl_tcp_mem配置为：“49152 65536 98304”。我们基于以下考虑选择相对较大的接收缓冲区：(1) 在现实世界中,系统管理员经常将 /proc/net/ipv4/tcp_rmem 配置为高,以适应高 BDP 连接。 (2) 我们想要演示将 /proc/net/ipv4/tcp_rmem 配置为高时给 Linux 系统带来的潜在危险。

4.1 接收环形缓冲区接收方

NIC 的接收环形缓冲区中的数据包描述符总数为384。正如第3 节所述,接收环形缓冲区可能是数据包接收的潜在瓶颈。我们的实验已经证实了这一点。在实验中,发送方向接收方传输 1 个 TCP 流,传输持续时间为 25 秒。实验结果如图9所示： ·正常情况下,接收环形缓冲区中只使用一小部分数据包描述符,并且使用的描述符会及时重新初始化和重新填充。 ·令人惊讶的是,可以看到在BL10 负载的少数情况下（@2.25s、@2.62s、@2.72s）,接收环形缓冲区中的所有384 个数据包描述符都被使用。此时,

更多传入数据包将被丢弃,直到重新初始化并重新填充所使用的描述符。经过仔细排查,发现BL10系统内存压力较大。在这种情况下,尝试分配新的数据包缓冲区来重新填充已使用的描述符

会失败,并且tR)(的速率为

r

实际上减少了;很快,接收环形缓冲区就用完了就绪描述符,并且数据包被丢弃。那些未能重新填充的已用描述符只能在Linux内核的页帧回收算法（PFRA)重新填充伙伴系统的空闲块列表后才能重新填充,例如通过收缩缓存或回收页框来自用户模式进程[12]。

在现实世界中,数据包丢失通常归咎于网络,尤其是 TCP 流量。很少有人意识到丢包可能经常发生在网卡上。

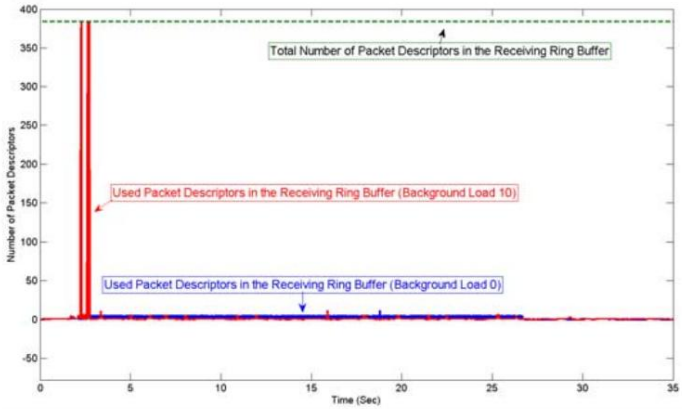

图 9 接收环缓冲区中使用的数据包描述符

4.2 TCP 和UDP在

TCP 实验中,发送方向接收方传输一个TCP 流,持续25 秒。

图 10 和 11 分别显示了背景载荷 (BL) 为 0 和 10 时的观察结果。 ·通常,预队列和乱序队列为空。积压队列通常不为空。数据包在测试网络中不会被丢弃或重新排序。然而,当数据包被 NIC 丢弃（图 9）或临时存储在积压队列中时,后续数据包可能会进入失序队列。 ·接收队列即将满。在我们的实验中,由于发送器比接收器更强大,因此接收器控制流量。实验结果证实了这一点。

·与图10 相比,图11 中的积压和接收队列显示出某种周期性。该周期与数据接收进程的运行周期相匹配。在图10中,对于BL0,数据接收过程几乎连续运行,但在BL10时,它以优先级循环方式运行。

在 UDP 实验中,发送方向接收方传输一个 UDP 流,持续 25 秒。

实验采用三种不同情况进行：(1)发送速率:200Mb/s,接收器后台负载:0； (2)发送速率:200Mb/s,接收方后台负载:10； (3)

发送速率:400Mb/s,接收器后台负载:0。图 12 和图 13 显示了 UDP 传输的结果。

·情况(1) 和(2) 均在接收器的处理限制范围内。接收缓冲区通常是空的。

·在情况(3) 中,接收缓冲区保持满。情况 (3) 表现出接收活锁问题 [8]。数据包在套接字级别丢弃。情况 (3) 的有效数据速率为 88.1Mbits,套接字处的丢包率为 670612/862066 (78%)。

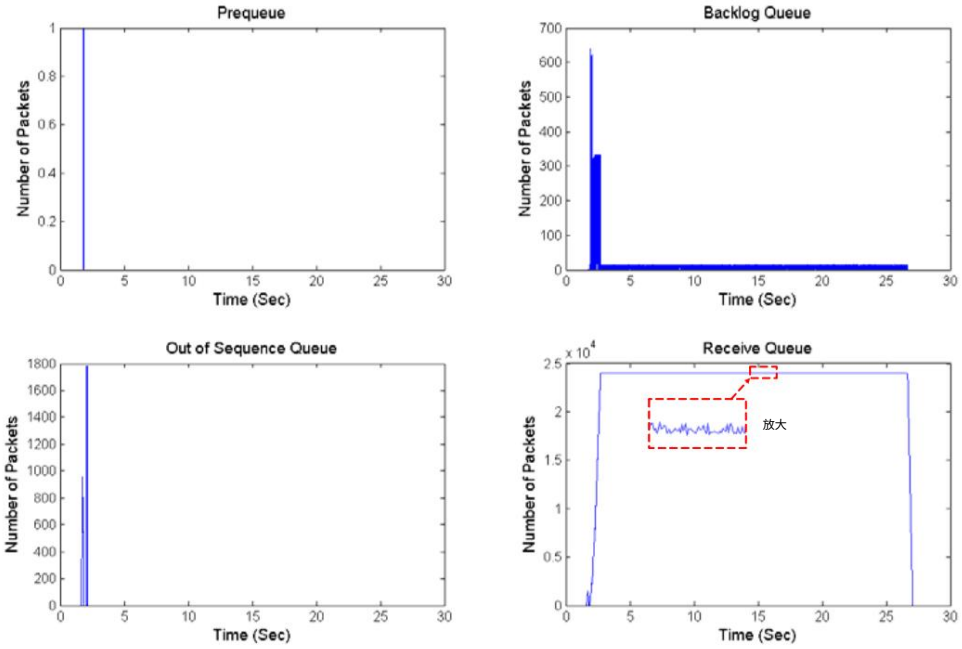

图 10 各种 TCP 接收缓冲区队列 – 后台负载 0

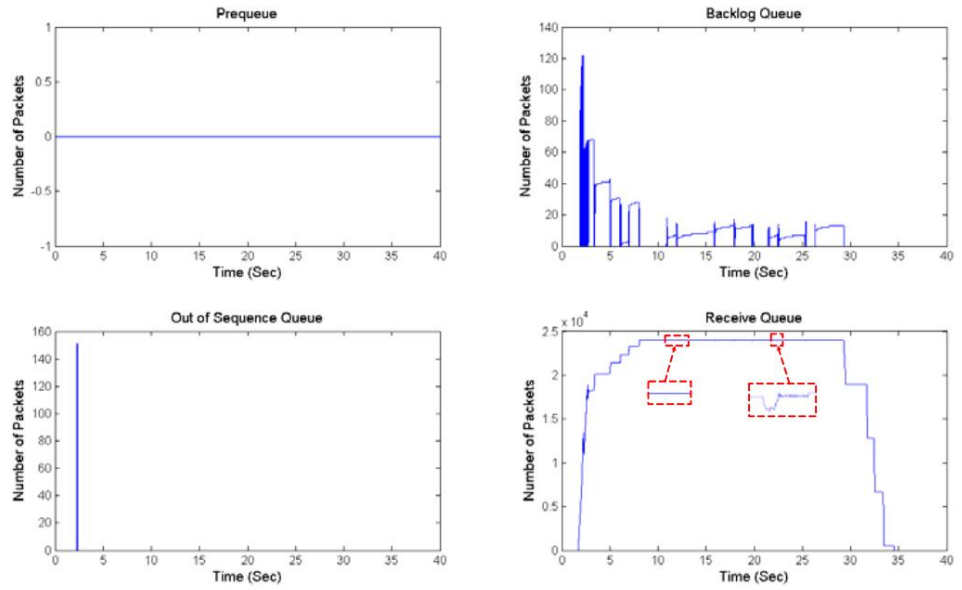

图 11 各种 TCP 接收缓冲区队列 – 后台负载 10

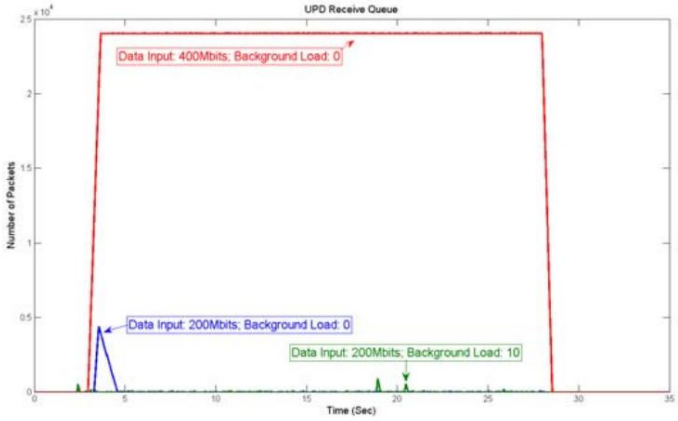

图 12 各种条件下的 UDP 接收缓冲区队列

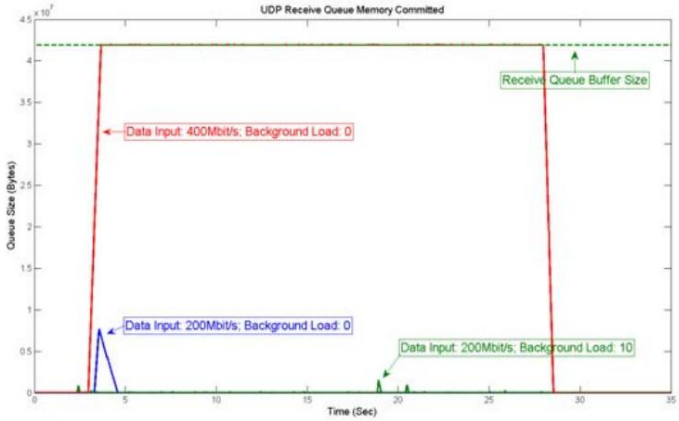

图 13 UDP 接收缓冲区提交内存

上述实验表明,当发送方比接收方快时,TCP（或UDP)接收方缓冲区将接近满。当套接字接收缓冲区大小设置得较高时,大量内存将被满的接收缓冲区占用（套接字发送缓冲区也是如此,这超出了本文的主题）。为了验证这一点,我们进行如下实验:发送方向接收方发送一个 TCP 流,持续 100 秒,所有进程都以 Nice 值 0 运行。我们记录 Linux 系统的 MemFree、Buffers、Cached,如/ proc所示/meminfo 在 50 秒处（根据上述实验,接收缓冲区在 50 秒处接近满）。这里， MemFree是系统中可用空闲内存总量的大小。Buffers和Cached分别是内存缓冲区高速缓存和页高速缓存的大小。当系统处于内存压力时,分配给缓冲区高速缓存和页高速缓存的页帧将被 PFRA [12][27]回收。另外,请注意： (1) 由于系统总内存为256MB,因此全部属于Lowmem。 (2) sysctl_tcp_mem配置为“49152 65536 98304”,这意味着最大 TCP 内存允许达到 384MB ‡

，如果可能的话。

实验结果如表3所示。可以看出,随着socket接收缓冲区大小的增加,系统的可用内存明显减少。具体来说,当

‡页面为 4K。

套接字接收缓冲区大小设置为170MB,缓冲区缓存和页面缓存都缩小了。为缓冲区高速缓存和页高速缓存分配的页帧由 PFRA 回收,为 TCP 节省内存。可以想象,如果socket接收缓冲区设置得很高,并且有多个同时连接,系统很容易耗尽内存。当内存低于某个阈值并且 PFRA 无法再重新声明页帧时,内存不足 (OOM) 杀手 [12] 开始工作并选择要终止的进程以释放页帧。严重的话甚至可能导致系统崩溃。

为了验证这一点,我们将套接字接收缓冲区大小设置为80MB,并同时运行五个与接收方的TCP连接,接收方很快就耗尽了内存并杀死了iperf进程。

	实验结果					
接收缓冲区大小 1M 200764 7300		10M	40M	80M	160M	170M
内存释放 (KB)	28060	189108	149056 95612 7384		3688	3440
缓冲区 (KB)		7316	7400 28112 28096		7448	2832
缓存 (KB)		28044			14444	6756

表 3 Linux 系统在不同接收缓冲区大小下的可用内存

显然,对于具有 256MB 内存的系统,允许整体 TCP 内存达到 384MB 是错误的。为了解决这个问题,我们将sysctl_tcp_mem重新配置为“40960 40960 40960”(最大TCP内存允许达到160MB)。

我们再次将套接字接收缓冲区大小设置为80MB,并重复上述实验。
无论有多少 TCP 连接同时流式传输到接收器,iperf 进程和接收器都可以正常工作。

以上实验表明,当/proc/net/ipv4/tcp_rmem(或/proc/net/ipv4/tcp_wmem)设置为高时,应相应配置/proc/net/ipv4/tcp_mem,以防止系统耗尽低内存。对于内存大于1G的IA-32架构Linux网络系统,我们建议将整体TCP内存限制为最多600MB。正如第3节所说,这是由于:IA-32架构的Linux网络系统通常采用3G/1G虚拟地址布局,kennel最多可以有896MB的Lowmem;内核代码及其数据结构必须驻留在Lowmem中,并且它们是不可交换的;为套接字接收缓冲区(和发送缓冲区)分配的内存也必须驻留在 Lowmem 中,并且它们也是不可交换的。当套接字接收缓冲区大小设置得很高并且有多个同时连接时,系统很容易耗尽Lowmem。总的 TCP 内存必须受到限制。

4.3 数据接收进程 下一个实验的
目的是研究数据接收进程的CPU 占有率变化时的整体接收性能。在实验中,发送方向接收方发送一个 TCP 流,传输持续时间为 25 秒。在接收端,数据接收进程的nice值和后台负载都是不同的。实验中使用的好值是:0、-10 和 -15。

Linux 进程的 Nice 值(静态优先级)范围为 –20 到 +19,默认值为零。 19 是最低优先级,–20 是最高优先级。好的值没有改变

由内核。 Linux 进程的时间片纯粹根据其nice 值来计算。进程的优先级越高,每轮执行接收到的时间片就越多,这意味着 CPU 份额越大。表 4 显示了各种好值的时间片。

不错的价值	时间片5 ms
+19	
	100毫秒
0	600毫秒
-10	700毫秒
-15 -20	800毫秒

表 4 尼斯值与时间片

图14 中的实验结果表明： ·数据接收进程的优先级越高,每轮执行接收到的CPU 时间就越多。较高的 CPU 份额意味着来自套接字接收缓冲区的实际数据包服务速率相对较高,从而改善端到端数据传输。

·后台负载越大,每轮完整执行所需的时间就越长。这降低了数据接收进程的运行频率及其整体CPU占用率。当数据接收进程的运行时间较少时,接收缓冲区内的数据的服务频率就会降低。此时TCP流控机制就会生效,限制发送速率,导致端到端数据传输速率下降。

实验结果证实并补充了我们在第 3 节中的数学分析。

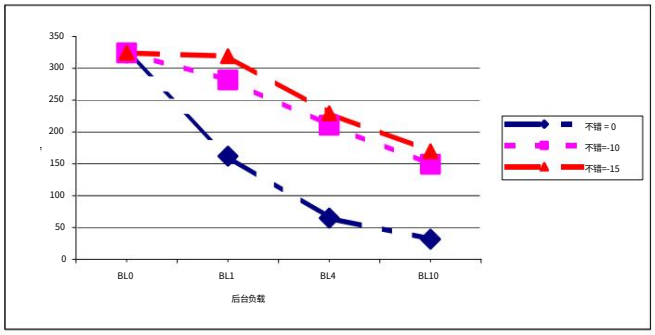

图 14 各种条件下的 TCP 数据速率

5. 结论

本文从网卡到应用程序详细研究了Linux系统的数据包接收过程。我们开发了一个数学模型来表征和分析 Linux 数据包接收过程。数学模型中,网卡和设备驱动接收过程用令牌桶算法来表示;其余的数据包接收过程被建模为排队过程

那些。

实验和分析表明,当接收环缓冲区中没有就绪数据包描述符时,NIC 可能会丢弃传入数据包。在过载的系统中,内存压力通常是导致网卡丢包的主要原因:由于Lowmem内存不足,无法重新填充已使用的数据包描述符;当所有处于就绪状态的数据包描述符都用完时,NIC 将丢弃更多传入的数据包。

实验和分析还表明,数据接收进程的CPU占用率是影响网络应用性能的另一个因素。在被数据接收进程消耗之前,接收到的数据包被放入套接字的接收缓冲区中。对于UDP,当套接字的接收缓冲区已满时,该套接字的所有传入数据包都将被丢弃。在这种情况下,对丢弃的数据包的所有协议处理工作

会被浪费。对于 TCP 来说,接收缓冲区已满会限制发送方的数据发送速率,导致 TCP 传输性能下降。为了提高数据接收进程的CPU占用率,可以采取以下方法:中断合并、巨型帧、TCP/IP卸载、减少所提供的系统负载、降低数据接收进程的nice值等。

对于 IA-32 架构的 Linux 网络系统,启用大套接字接收（发送)缓冲区大小时应更加注意:将 /proc/net/ipv4/tcp_rmem（或 /proc/net/ipv4/tcp_wmem)配置得较高高 BDP 连接的性能,但系统可能会耗尽 Lowmem。因此,应该对/proc/net/ipv4/tcp_mem进行相应配置,以防止系统耗尽Lowmem。

我们研究了网络和 CPU 速度按照当今标准来说适中的系统,以便处理成熟的整体系统设计。我们希望我们的结果能够随着系统所有部分的速度扩展而保持不变,直到并且除非对数据包接收过程进行一些根本性的改变。

参考

[1] H. Newman等人， “超轻型项目:网络作为集成和托管资源
数据密集型科学”。科学与工程计算,卷。 7、没有。 6,第 38 – 47 页。

[2] A. Sim等人， “DataMover:通过广域网实现强大的 TB 级多文件复制”，
第 16 届国际科学和统计数据库管理会议论文集,2004 年,第 403 – 412 页。

[3] D. Matzke，“物理可扩展性会破坏性能提升吗？”计算机,卷。 30,没有。 9,1997 年 9 月,第 37 – 39 页。

[4] D. Geer，“芯片制造商转向多核处理器”,计算机,卷。 38,没有。 5,2005 年 5 月,第 11 – 13 页。

[5] M. Mathis等人， “Web100:用于研究、教育和诊断的扩展 TCP 仪器”，
ACM 计算机通信评论,卷。 33、没有。 3、2003 年 7 月。
T. Dunigan等人， “TCP 调优守护进程”,SuperComputing 2002。 [6]

[7] M. Rio等人， “Linux 内核 2.4.20 中的网络代码图”,2004 年 3 月。 [8]
JC Mogul等人， “消除中断驱动内核中的接收活锁”,ACM Transactions on Computer Systems,卷。 15、没有。 3,第 217--252 页,1997 年。

[9] K. Wehrle等人， Linux 网络架构 – Network Pro 的设计和实现
Linux 内核中的 tocols,Prentice-Hall,ISBN 0-13-177720-3,2005。 [10]

www.kernel.org。

[11] R. Love,Linux 内核开发,第二版,Novell Press,ISBN:0672327201,2005。

[12] DP Bovet,了解 Linux 内核,第三版,O Reilly Press,ISBN:0-596-00565-2，
2005年。

[13] J. Corbet等人， Linux 设备驱动程序,第 3 版,O Reilly Press,ISBN:0-596-00590-3,2005 年。

[14] AS Tanenbaum,计算机网络,第三版,Prentice-Hall,ISBN:0133499456,1996。

[15] AO Allen,概率、统计和排队论与计算机科学应用,第 2 期
版,学术出版社,ISBN:0-12-051051-0,1990。

[16] Y. Hoskote等人，“采用 90 nm CMOS 的 10 Gb/s 以太网 TCP 卸载加速器”,IEEE Journal
固态电路,卷。 38,没有。 2003 年 11 月 11 日,第 1866 – 1875 页。

[17] G. Regnier等人， “数据中心服务器的 TCP 加载”,计算机,卷。 37、没有。 2004 年 11 月 11 日，
第 48 – 58 页。

[18] D. Freimuth等人， “服务器网络可扩展性和 TCP 卸载”,2005 年 USENIX 年度技术会议论文集,第 209--222 页,加利福尼亚州阿纳海姆,2005 年 4 月。

[19] J. Mogul，“TCP Offload is a Dumb Idea Whose Time has Come”,第九届操作系统热门话题研讨会论文集,夏威夷利胡埃,2003 年 5 月。

[20] DD Clark等人， “TCP 处理开销分析”,IEEE 通信杂志,卷。 27、没有。 2,1989 年 6 月,第 23 – 29 页。

[21] K. Lin等人， “RED-Linux 中实时调度程序的设计和实现”,论文集
IEEE,卷。 91,没有。 7,第 1114-1130 页,2003 年 7 月。
[22] S. Makineni等人， “商业服务器工作负载中 TCP/IP 数据包处理的性能特征”,IEEE 国际工作负载特征研讨会,2003 年
10 月,第 33 – 41 页。

[23] Y.Yasu等。 al.， “Event Builder 的千兆位以太网服务质量”,IEEE 核科学研讨会会议记录,2000 年,第 1 卷。 3,第
26/40 - 26/44 页。
[24] J. Silvestre等人， “多媒体应用现场总线中使用大帧尺寸的影响”,第 10 届 IEEE 新兴技术和工厂自动化会议,卷。 1，
第 433 页-
440,2005 年。
[25] 传输控制协议,RFC 793,1981。 [26] www.linux-mm.org。

[27] M. Gorman,了解 Linux 虚拟内存管理器,Prentice Hall PTR,ISBN:0131453483,2004 年 4 月。

[28] CS Rodriguez等人， Linux(R) 内核入门:x86 和 PowerPC 的自顶向下方法
建筑,Prentice Hall PTR,ISBN:0131181637,2005。
[29] L. Arber等人， “消息缓冲区对齐对通信性能的影响”,Paral
lel 处理信件,卷。 15、没有。 1-2,2005 年,第 49-66 页。
[30] K. Chen等人， “提高 Intel Itanium 架构上的企业数据库性能”,Linux 研讨会论文集,2003 年 7 月 23-26 日,加拿大
安大略省渥太华。
[31] http://dast.nlanr.net/Projects/Iperf/。
[32] J. Semke等人， “自动 TCP 缓冲区调整”,计算机通信评论,ACM SIGCOMM,卷。 28、没有。 1998 年 10 月 4 日。