为了低延迟和高性能而重新架构的数据中心网络和堆栈

摘要

现代数据中心网络提供非常高的容量通过冗余的Clos拓扑和低交换延迟，但是传输协议很少能提供与之相匹配的性能。

这里介绍NDP这个新的协议，和现有的其他传输协议相比，实现了近乎最优的成功次数，在各种脚本中的短传输和高流量模型下，包括incast模式下。NDP切换缓冲区是非常浅的，当他们填充这些切换时，以修整的数据包给头部位置并且优于之前的头部，这给了接受者一个来自所有发送者即时需求的完整视角，这一点，室在RTT周期中，我们能够应对来自不同发送者的优先传输处理和其他请求的协议基础。我们在linux主机上在一个软件交换机上，在一个基于NETFPGA的硬件交换机上，以及在P4上使用DPDK实施NDP。在我们的实现和大规模的模拟当中我们评估了NDP的表现，同时证明它支持了低延迟和高吞吐。

CCS 概念

网络→网络协议；数据中心网络；

关键词

数据中心；网络栈；传输协议

1 介绍

近几年，数据中心发展迅速，随着CLOS拓扑越来越普及，以及一个新的关注点在低延迟上，在聚合以太网V2上第一次使用升级的传输协议诸如DCTCP，以及最近像RDMA这样的方案，在交换机上使用了以太网流控制来避免因为拥塞丢失的数据包。

在一个低负荷以太网中，以太网流控制能给主导数据中心负载请求/响应流提供非常好的低延迟表现，数据报是在排队而不是丢失，如果有必要产生反向压力，暂停转发，这样就不会有时间浪费在过度保守的启动或者等待超时重传上。然而，基于部署RoCE v2在微软的经验上，Gau等人指出低丢失的网络并不能保证低延迟。他们总结“对于RDMA来说如何同时保证低网络延迟和高的网络吞吐率仍然是一个开放的问题”。

在这篇论文中我们描述一个新的数据中心协议架构，NDP（采取了一种不同的方法来同时达到低延迟和高吞吐率）。NDP没有连接握手，允许流以全速率立刻开始发送。我们使用逐个报文的多路负载均衡（以重新排序为代价避免了核心网络拥塞），并在交换机中使用一个类似CUT PAYLOAD的方法（当一个交换队列填入时，削减数据报的负载）。这使得网络元数据无损，但流量负载并非如此。尽管重新排序，无损元数据给接受者一个完整的图像关于流入流量，并且我们利用它来建立一个基础的新的传输协议（对断流量来说达到低延迟），甚至在逻辑路线流量模型中，流通往不同目的地的流之间是达到最小的冲突。

我们已经部署NDP在LINUX主机、在一个软件交换机、一个基于NETFPGA SUME的硬件交换机、在P4、以及在仿真理，我们将证明NDP能够做到：

1.比DCTCP或DCQCN有更好的短流表现。

2. 只有8个数据报的交换队列却在高负载情况下比95%的大型网络表现好。

3. 在Incast案例近乎完美的低延迟和公平。

4.通往不同主机的流之间的最小的干扰。

5.在incast期间掉队流量的有效优先考虑。

2 设计空间

内部数据中心网络流量主要由请求/响应类似RPC的协议组成。意味着网络利用率很少非常高，但是应用是极具爆发性的。这个大的问题如今是延迟，尤其对于类似RPC的短工作负载，以线程头部堵塞为代价，现在的应用经常重用TCP连接通过多次请求来摊销TCP握手的延迟代价。

提升协议栈这么多是否可能（每个请求能够使用一个新的连接并且同时期望接近底层网络的原始延迟和带宽，甚至在高负载下）？我们将展示这些目标是可以达到的，但是为了达到这样牵涉到变化例如交通如何路由，交换机如何处理过载，以及最重要的，需要一个完全不同于今天使用的传输协议。在我们在第三章描绘我们的解决方案之前，我们首先强调关键的必须被考虑的结构点。

2.1 端到端的服务需求

应用需要来自一个数据中心网络的什么？

独立的位置。一个分布式应用的元素们运行在数据中心的哪个机器上不重要。这一点普遍被达成了在使用高容量平行CLOS拓扑结构。这种拓扑结构有充分的截面带宽以至于核心网络不是瓶颈。

低延迟。CLOS网络能提供带宽，建模来平衡路线间的负载，但经常在提供低延迟服务上达不到要求。可预言的非常低延迟请求/响应表现是关键的应用需求，但是它是最难满足的。这比大文件传输表现更重要，尽管高的吞吐仍然是一个需求，尤其对于存储服务器。这个策略必须首先为低延迟优化。

incast。数据中心负载经常需要发送请求给大量工人然后处理他们的几乎同时的反应。引起一个叫做incast的问题。一个好的网络栈应该保护应用规避incast交通模型的负面影响当提供低延迟时。

优先级。对于一个接受者来说同时处理一些来的流对应于不同的请求是很普遍的。例如，可能发出两个不同的请求给工人，然后对这些请求的响应现在和最后一个响应第一个请求的一同到达（覆盖了对于第二个请求的第一个响应）。一些应用在他们开始前需要所有一个请求的所有响应。一个令人满意的性能对于接受者来说是能够使掉队的流量优先。接受者是唯一的实体（能够动态优先他的进入流量），而且这影响到协议设计。

2.2 传输协议

目前的数据中心传输协议满足了这些应用需求的一部分，但满足所有需要数据中心传输协议的一些不一般的能力。

**0-RTT连接建立**。为了最小化延迟，一些应用对于小的对外传输喜欢0-RTT传输（或者1-RTT的请求/响应）。我们需要一个协议（在发送数据前不需要握手来完成），但这会造成安全和正确性问题。

**快速开始。**另一个0-RTT传输的意义是传输协议不能寻找可用的带宽-为了最小化延迟，必须假设带宽是可用的，乐观的发送一个满的原始窗口，然后当不是这样是适时的重新启动。与互联网形成对照，更简单的解决方法在数据中心环境中是可能的，因为链路速度和网络延迟（除了排队延迟）能够提前预知。

**每个数据包ECMP。**CLOS拓扑的一个问题是路线上流量的每个ECMP哈希的流能引起未知的流冲突；一个部署发现这减少了40%的吞吐。对于大的传输，多路线协议例如MPTCP能够建立足够的分支流来找到未使用的路线，但是他们鲜能在短传输的延迟问题上发挥作用。唯一的解决方法是将每个路线加在每个包的基础上。这使得传输协议的设计复杂化了。

**容忍重排序握手。**如果我们在Clos网络中使用每包多路径转发执行零RTT传输，则甚至第一波数据包可能以随机顺序到达。 这种影响会影响连接设置：第一个到达的数据包将会不是连接的第一个数据包。 这样的传输协议必须能够建立连接状态，无论哪个来自初始窗口的数据包首先到达。

**为Incast优化**。虽然Clos网络的核心容量配置良好，但当应用程序同时向许多工作人员发出请求时，incast流量可能会使任何传输协议都变得很困难。这种流量模式会导致高丢包率，特别是如果传输协议在第一个RTT中比较激进。较好的处理这个需要交换机的一些帮助。

2.3交换服务模型

应用需求，传输协议和网络交换机的服务模型紧密耦合，需要进行全面优化。特别相关的是当交换机端口拥塞时会发生什么。交换机服务模式严重影响协议和拥塞控制算法的设计空间，并与转发行为紧密耦合：每个数据包多路径负载平衡是最理想的，因为它可以最大限度地减少热点，但它使端系统推断网络拥塞的能力复杂化，增加了优雅超载行为的重要性。

作为拥塞反馈机制的损失具有下列优点：丢弃的数据包不使用瓶颈带宽，并且丢失仅影响穿过拥塞链路的流量 - 并非所有方案都具有这些属性。 缺点是它导致了不确定性一个数据包的结果。重复或选择性ACK来触发重传仅适用于长期流动。 短流，尾丢失是常见的，然后你必须回退重传超时（RTO）。如果你能限制这些，短暂的RTO只是安全的

网络延迟，所以你需要维持短队列[4]，这反过来限制了你可以使用的拥塞控制方案。 丢包也严重地与每分组多路径转发耦合; 因为流的数据包无序到达，所以丢失检测非常复杂 - 快速重新传输通常是不可能的，因为对于一个数据包来说，整个窗口无序地到达数据包并不罕见。

ECN帮助显着。 DCTCP使用ECN [32]数据包标记的阈值以及拥塞控制方案旨在推进标志制度。这大大减少了损失长寿命的流量，并允许使用小缓冲区，减少排队延迟。尽管如此，ECN的收益较低因为流量没有时间对ECN反馈做出反应。在练习开关与ECN一起使用大型共享缓冲区这会减少incast的损失，但重传定时器必须减少ECN明显的帮助。 DCTCP使用ECN [32]数据包标记的阈值以及拥塞控制方案旨在推进标志制度。这大大减少了损失长寿命的流量，并允许使用小缓冲区，减少排队延迟。尽管如此，ECN的收益较低因为流量没有时间对ECN反馈做出反应。在练习开关与ECN一起使用大型共享缓冲区这减少了失败，但重发定时器必须更少。使用802.3X暂停帧[23]或基于802.1Qbb优先级的流量控制（PFC）[24]的无损以太网可以防止丢失，避免在协议中需要积极的RTO。在低利用率时，这可以有效地实现低延迟—一个突发将会到达链路可以转发的最大速率，而不需要等待重传。问题出现在分层拓扑中更高的利用率，这种情况发生在同一个传出的哈希端口，并且在802.1Qbb的情况下使用相同的优先级，可能导致传入端口被暂停。这会对其他人造成附带损害流经相同的输入端口，输出不同的输出端口。 随着大型浓缩酒会，暂停可以逐渐向核心后退开关。 无损以太网也与每个分组的多路径转发严重交互，因为不同的交换机可能会暂停不同的交换次数，加剧重新排序并使终端系统设计复杂化。

削减有效负载（CP）[9]试图在没有损失的情况下获得无损的好处。 它丢弃数据包有效载荷，但不丢包头部，减轻过载，同时避免数据包的不确定性结果。 它显示出很大的希望，但有两个问题。 第一，在严重超载的情况下，它容易发生拥挤塌陷只有标题被转发。 其次，因为标题已排队以FIFO的方式，尾部“损失”至少需要一个RTT。 另外，CP，如最初所建议的那样，对每个流使用单路径转发。

3.设计

我们的主要目标是短流量的低完成延迟以及更长流量的可预测高吞吐量。为了完全实现这些目标，NDP会影响整个栈，包括交换机行为，路由和全新的传输协议。我们以一个简短但简化的设计原理来领导，以展示图1中的部分如何组合在一起，然后详细介绍本节的其余部分。

clos拓扑结构在内核中具有足够的带宽以满足所有需求，只要它具有完美的负载平衡。为了避免流量核心链路上的冲突（影响延迟和吞吐量），对每条流量在多条路径上进行负载平衡至关重要。 平衡短的流量需要每个数据包的多路径负载均衡，但不可避免的数据包将被重新排序。

为了实现最小的短流延迟，发送者在发送之前无法进行探测：他们必须以线路速率发送第一个RTT，这在大多数情况下运作良好。当发送者执行每个数据包的多路径负载平衡时，如果以线路速率发送引起拥塞，那便因为多个发送者正在发送到同一个接收者。即使那样，接收器的链路也完全被占用，所以这本身不是问题。

为了保证低延迟，交换队列必须很小。这意味着碰撞的流量会溢出队列。 数据包丢失与多路径重新排序相结合使得无法推断发生了什么并且足够快地重新传输以避免影响延迟; 这违反了低延迟目标。完全防止丢包增加排队延迟; 如果这是通过暂停入栈流量完成的，如使用无损以太网，这会影响其他不相关的流量，违反其低延迟和可预测的高吞吐量目标。我们寻求一个丢包和无损之间的中间地带。

数据包修整与CP所执行的类似，就是这样一个中间地带。交换机队列可以很小，接收机仍然可以通过检查收到的剪切头来发现发送了哪些数据包。然而，为了最小化重传延迟，修剪报头和控制包需要优先。到达修剪头部会准确告诉接收器需求是什么，所以通过使用接收器拉出的协议，接收器可以精确控制传入的流量。 这避免了持续的过载并允许更多重要的数据包首先被接收方自行决定。

3.1 NDP交换服务模型

使用CP时，当交换机的队列填充超出固定阈值时，不是丢弃数据包，交换机会修剪数据包有效载荷，排队只是头。 基本原理是数据包不会悄悄丢失，无需等待即可快速重新超时传输。由于数据中心网络中的距离很短，这种重传可以很快到达。

除了交换更改外，CP还提议对TCP进行微小更改以提高incast性能。我们希望超越CP，并使用数据包修整作为非常积极的网络架构的基础，专注于非常低延迟的服务。 但是，如果使用普通的CP，会出现几个问题。

首先，CP可能遭受一种形式的拥塞崩溃。图2显示了当数据包以比出站链路支持的速率高得多的速率到达交换机时发生的情况。 许多无响应流程汇聚在一个只支持其中一个的10Gb / s链路上，就像在极端的服务器聚合场景中一样。该图显示实现的理想公平份额生产量的百分比。CP流量的平均吞吐量减少，因为链路的增加部分被修剪的数据包报头占用。 该图显示了CP的最佳情况，包含9KB 超大包。对于1500字节的数据包，崩溃速度要快得多。

其次，数据中心网络非常规则，因此可能发生相位效应[14]，从而导致不公平的吞吐量。图2中的虚线曲线显示了表现最差的10％流量的平均吞吐量。尽管我们注意到相位效应会使CP非常不公平该图显示了模拟结果; 真实世界的相位效应有时可以通过数据包的时间变化来减少操作系统调度导致的传输。

最后，CP旨在提供数据包丢失的低延迟反馈。但是，由于CP使用FIFO队列，因此只有在接收到所有前面的数据包后才能发送反馈，导致重传被引发之前的延迟。我们希望在交换机中运行非常小的缓冲区，并拥有其中一个队列并且在队列有机会流失之前重新传输。 这在FIFO排队中是不可能的。

NDP交换机对CP进行了三项主要更改。 首先，NDP交换机维护两个队列：数据包的优先级较低的队列，以及修剪头，ACK和NACK的较高优先级队列1。这看起来可能不符合直觉，但它提供了一个数据包没有做到的最早可能的反馈，通常在违规队列甚至有时间耗尽之前允许重传到达。这可以提供至少与无损以太网一样好的低延迟行为，而不会造成暂停导致的附带损害。

其次，交换机在高优先级“头部队列”和较低优先级的“数据包队列”之间执行加权循环。以10：1的报头与数据包的比率，这可以实现早期反馈，而不会受到拥塞崩溃的影响。

最后，当数据包到达并且低优先级队列满时，交换机以50％的概率决定是否修剪新到达的数据包，或者低优先级队列尾部的数据包。这打破了相位效应。图2显示了NDP开关如何避免CP的崩溃，并且还避免了强烈的相位影响。

3.1.1 路由

我们希望NDP交换机执行每个分组的多跳转发，以便在所有并行路径上平均分配流量突发源和目标之间可用。这可以通过至少四种方式来完成：

•执行每个数据包的ECMP; 开关随机选择下一个为每个分组跳转;

•明确源路由流量;

•使用标签交换路径; 发件人选择标签;

•目的地址指示要采取的路径; 发件人在目标地址之间进行选择；

为了负载平衡的目的，后三者是等价的 - 发送者选择一条路径 - 他们在发送者表达方式上有所不同路径。我们的实验表明，如果发送者选择路径，它们可以比随机切换更好地实现负载平衡选择路径。这允许使用稍小的开关缓冲器。

与互联网不同，在数据中心，发件人可以知道拓扑，因此知道有多少路径可用于目的地。每个NDP发送者都将目录的路径列表随机排列，然后按此顺序在路径上发送数据包。 之后在每个路径上发送一个数据包，它会再次随机排列路径列表，并重复该过程。 这可以在所有路径上平均分发数据包，同时避免两个发件人之间无意中的同步。 这种负载平衡对于实现非常低的延迟非常重要。 如果我们使用非常小的数据包队列（只有八个数据包），我们的实验表明，这个简单的方案可以增加网络的最大容量在每包随机路径选择上高达10％。

根据网络是L2还是L3交换，可以使用标签交换路径或目标地址进行选择一条路径。例如，在一个L2 FatTree网络中，只需要将每个核心交换机设置为标签交换路径L2地址从那里接管，因为FatTree只有一条从核心交换机到每个主机的路径。在L3 FatTree中，每个主机都会获得多个IP地址，每个核心交换机一个。 通过选择目标地址，发送方选择核心交换机一个数据包遍历。

3.2传输协议

NDP使用专门设计的接收器驱动的传输协议来利用多路径转发，数据包修整和短切换队列。 每个步骤的目标首先是尽量减少短期转移的延迟，然后为更大的转移最大化吞吐量。

在启动连接时，传输协议可能会比较悲观，例如TCP，并且假设有最少的备用网络容量。TCP在三次握手完成后开始发送数据，最初拥有一个小的拥塞窗口[11]，然后将其加倍每个RTT直到它填满管道。缓慢启动适用于互联网，其中RTT和链路带宽按指令顺序不同数量巨大，而且更具攻击性的后果严重。但是，在数据中心中，链路速度和基准RTT更加均匀，并且可以事先知道。而且，网络利用率通常相对较低[7]。在这样的网络中，为了使延迟最小化，我们必须保持乐观，并假设在连接的第一个RTT中将有足够的容量发送完整的数据窗口而无需探测。如果交换缓冲区很小，那么在低延迟数据中心环境中，整个窗口很可能只有一个大约12个数据包给出了光速等待时间和跳数。

但是，如果发现容量不足，数据包将丢失。使用正常的传输协议，每个数据包多路径转发的组合以及在第一个RTT中具有侵略性是混淆的秘诀。一些数据包到达，但在一个随机顺序，有些则不。不可能迅速地分辨出实际发生的事情，因此发件人必须回避保守态度重新传输超时以纠正这种情况。

增加交换机缓冲可以在一定程度上缓解这种情况，但会增加延迟，但不能防止丢失大型的浓烟。ECN也不能用积极的短期流量来防止损失。暂停帧可以防止丢失，并且可以帮助显着在这里，但是我们会在第6.1节中表明，这会给延迟到不相关的流程带来自己的重大问题。

这是NDP交换机中的数据包修整真正进入其中的地方。 经过修剪的数据包到达接收器的头部消耗的带宽很少，但是会通知接收器准确地发送了哪些数据包。 在推断发生的事情时，数据包到达的顺序并不重要。 优先排队确保这些报头很快到达，并且返回给发送方的诸如NACK的控制分组快速到达; 确实足够快以引起在溢出队列已经耗尽之前到达的重传，所以链路不会空闲。

这在图3中示出。在时间ttrim，来自九个不同源的分组几乎同时到达ToR交换机。到目的地链路的八个分组队列填满，并且来自源9的分组被修剪。 数据包1完成转发后，数据包9的报头得到优先处理。在头部它到达接收器，它产生一个NACK包2。分组9在trtx被重新发送并到达ToR在队列7仍然被转发时切换队列。链接到目的地永远不会空闲，并且数据包9到达tarrive，如果PFC阻止了它的损失的话，那也是同样的时间暂停上游开关。

在采用每分组多路径的克洛斯拓扑结构中，可以构建的唯一热点是来自多个源的流量汇聚一个接收器。使用NDP，修剪头显示接收器的精确需求; 它确切地知道哪些发件人想要发送哪些数据，所以最好决定在连接的第一个RTT之后要做什么。以线速发送完整的数据窗口后，NDP发件人停止发送。从那时起，该协议是接收器驱动的。NDP接收器请求来自发件人的数据包，调度发送这些请求以便它们引出的数据包到达速度与接收者的链接速度相匹配。请求的数据可以是重新整理的数据包，也可以是新数据其余的转移。该协议因此如下工作：

•发件人发送完整数据窗口而不等待回复。数据包携带数据包序列号。

•对于每个到达的报头3，接收方立即发送NACK通知发送方为重发准备数据包（但尚未发送）。

•对于每个到达的数据包，接收方立即发送ACK以通知发送方数据包已到达，因此缓冲区可以被释放。

•对于每个到达的报头或数据包，接收方都会在其拉入队列中添加一个PULL数据包，并在适当的时候发送给相应的发送方。一个接收器只有一个拉式队列，由它作为接收器的所有连接共享。

•PULL数据包包含连接ID和每个发件人拉式计数器，该数量在发送给该发件人的每个PULL数据包上递增。

•接收方从每个接口的拉取队列中发出PULL数据包，调整它们从发送方引出的数据包然后到达接收方的链路速率。从不同的地方拉包默认情况下，连接的服务是公平的，或者当流量具有更高的优先级时，严格区分优先级

•当一个PULL数据包到达发件人时，发件人将发送许多数据包随着拉计数器的增加而增加。 任何数据包首先发送排队等待重发，然后发送新数据。

•当发送者用完数据发送时，它会标记最后一个数据包。

当最后一个数据包到达时，接收方从其拉取队列中删除该发送方的任何拉数据包，以避免发送不必要的拉数据包。发件人稍后想要的任何后续数据发送将被推送而不是被拉。

由于数据包修剪，数据包实际上丢失是非常罕见的; 通常这是由于腐败。由于ACK和NACK是立即发送，优先转发的，并且所有交换机队列都很小，发送方可以很快知道数据包是否真的丢失。在10Gb / s FatTree拓扑结构中，八个分组交换队列，9KB jumbogram和存储转发交换机，每个分组需要7.2μs的序列化。考虑到NDP的优先级排队，最差的网络RTT大约为400μs，典型的RTT要短得多。这允许使用非常短的重传超时来为这种损坏的分组提供可靠性。

PULL数据包的作用类似于TCP的ACK时钟，但通常与ACK分离，以便无需调整影响重传超时机制。例如，在一个大型聚合场景中，在起搏器允许发送它们之前，PULL可能在接收器的拉队列中花费相对较长的时间，但是我们不希望延迟ACK，因为这样做需要在重发时更保守超时。

紧急行为是推动连接中的第一个RTT数据，并且拉动数据的后续RTT以达到接收机的线路速率。 在incast的情况下，如果许多发送者同时发送，他们的许多第一个数据包窗口将被修剪，但随后接收器拉动确保聚合来自所有发送者的到达速率与接收者的链路速度相匹配，很少或没有数据包被修剪。

3.2.1 应付重新排序

由于每个数据包的多路径转发，这两个数据都是正常的数据包和反向路径ACK，NACK和PULL要重新排序。基本的协议设计对重新排序很有效，因为它不需要从其他数据包的序号进行推断。但是，重新排序仍然需要考虑。

尽管PULL数据包是优先级排队的，但它们并不抢占数据包，所以在不同路径上发送的PULL数据包经常出现的顺序，增加了重发的突发性。为了减少这种情况，PULL带有一个拉序列号。 接收器为每个连接都有一个单独的拉序列空间，每个连接增加一个拉送。在收到PULL后，发送方随着拉取序列号的增加而发送尽可能多的数据包。例如，如果PULL延迟，则发送的下一个PULL可能首先经由不同的路径到达，并且将拉出两个分组而不是一个。这可以稍微减少突发事件。

3.2.2第一个RTT

与在TCP中，在数据交换之前发生SYN / SYN-ACK握手不同，我们希望NDP数据在第一个RTT中发送。 这增加了三项新要求：

•对欺骗源IP地址的请求具有强大的功能。

•确保没有连接被无意中处理了两次。

•应对第一个RTT内的多路径重新排序。

T / TCP [8]和TCP Fast Open [10]都扩展TCP以在第一个RTT中发送数据。TFO通过提供给定的令牌来防止欺骗

由上一次连接中的服务器执行，但不能确保最常用的语义。 T / TCP使用单调递增的连接ID来提供最多一次的语义，但它对欺骗不健壮。如果SYN不是第一个到达的数据包，那么它们都不能很好地处理。

新民主党的要求略有不同。虚拟机管理程序或NIC中可以阻止欺骗，或者将VXLAN用于多租户数据中心，所以这不是一个大问题。尽管语义是最重要的，但是T / TCP的解决方案对于背对背短连接之间的多路径重新排序并不稳健。 NDP通过在客户端和服务器端都保持时间等待状态来解决这个问题，因此可以拒绝重复连接。 作为最大段寿命在1ms以下，额外状态量相当小。

最后，可能会在第一个RTT中发送多个数据包，但是首先到达的往往不是第一次发送的。 每个数据包都要健壮在第一个RTT中携带SYN标记以及其偏移量序列号来自连接中的第一个数据包。 这允许连接状态由哪个数据包首先到达建立

3.2.3鲁棒性优化

如果网络行为正常，上述协议表现良好。但是，有时链路或交换机会失败。 这通常由路由协议检测，然后路由失败。 NDP数据包是源路由的，因此NDP主机也需要接收这些路由更新以了解哪些路径可以避免。 但之前路由协议已通知大家，到达故障链路的数据包将丢失。其他更微妙的失败也是可能的，例如10Gb / s链路决定协商到1Gb / s，导致热点无法立即通过路由检测到。以前工作表明，使用单路径拥塞控制（例如TCP）和网内分组喷洒会导致这些场景的性能大幅下降，因为传输协议并不知道其路径中只有一条路径行为不当[12,31]

NDP结合了可在这种情况下大大提高性能的优化。 发件人记录路径和发件人跟踪每个数据包经过的路径。 当一个ACK或NACK到达数据分组路径的ACK或NACK计数被发送过来增加。通常情况下，Clos拓扑正在运行NDP，所有路径的ACK与NACK的比率应该非常相似。但是，如果失败导致了不对称，一些链接将会有过多的NACK计数。 当发件人将其排列在路径列表中时临时删除路径集中的异常值

丢包应该几乎不会发生。重发丢失数据包的NDP发送方总是在不同的路径上重新发送它。 一条路径每丢失一个数据包，丢失计数器也会增加。任何与数据包丢失相关的异常路径也是暂时的从路径集中删除。

这些机制允许NDP对于路径不再具有类似性能的网络（无论出于何种原因）具有鲁棒性性能损失最小。 依赖于流量ECMP多路径转发的传统协议比较困难失败，并依靠路由来检测和避免不良路径。

3.2.4 返回发送者

数据包修整可以应付大型百慕大群岛，而无需删除任何头文件。然而，非常大的浓度可能会超过流量头排队，造成损失。丢失的数据包将被重新发送迅速，当发件人的RTO到期。小队确保最大RTT为400μs，最大RTO可以安全地降低作为1ms。在incast期间，架顶式交换机队列可以容纳八个9KB数据包和（在相同数量的内存中）1125 64字节

溢出之前的标题。接收器将会推送重传在这1125个数据包中，为了保持其链路饱和，对其进行了调整。在10Gb / s时，925个11KB的数据包将占用接收器的链路8ms，因此任何由于RTO而重新发送的数据包仍然可以到达队列在链路闲置之前超过流量。

但是，有时整个传输将适用于单个数据包，并且该传输可能具有高优先级 - 例如，它可能来自先前请求的失散者。 如果这样的数据包丢失，依靠在RTO上增加了不必要的延迟。 作为一个优化，当头部队列溢出，交换机可以交换发送者和

收件人的地址，并将标题返回给发件人。 发件人然后可以重新发送违规的数据包。但是，总是重新发送可能会导致原始的incast的回声。只有NDP才会重新发送不期待更多的PULL - 即没有数据包被确认或NACK但尚未拉，或如果从第一个所有其他数据包窗户也被退回。这避免了回声回声，但保持了回声拉时钟去。 如果大多数数据包最近都有，NDP也会重新发送确认而不是NACK - 这表示一个不对称的网络，

在不同的工作路径上重新发送是有道理的。

返回发件人是一种优化; 根据我们的经验，它只能在非常大的浓缩酒中投入使用。 本质上在Clos拓扑中使NDP对元数据无损失; RTO仅在数据包损坏或发生故障时触发。

图4显示了432节点FatTree仿真的结果演示了这些机制的实际应用，给出了延迟的CDF从首次发送数据包到Ack发送时，包括因重传造成的任何延迟。置换曲线显示每个主机何时从另一个主机发送和接收数据随机显示每个主机发送到随机主机 - 两者都有这些情况完全加载FatTree，但中等延迟仍然存在大约100μs。 Incast曲线显示100时发生的情况节点同时发送到单个节点;它们的大小不同转移。在135KB的情况下，所有节点首先发送整个文件RTT;这不仅会导致较高的调整率，而且还会导致较高的调整率头部队列中，25％的头部被返回给发送者。尽管如此，最后一个数据包以11,055μs的速度到达，仅比2％晚理论最佳到达时间。在1350KB的情况下，第一个数据窗口的时间与135KB的传输时间相同，但是转移的其余部分顺利进行，无需修整和a中位潜伏期为95μs。

拥塞控制

精明的读者现在可能想知道NDP的作用了拥塞控制。 答案很简单：NDP在克洛斯拓扑中不执行任何拥塞控制。 正如我们将要表明的，恰当的组合对于拥塞控制来说是不必要的

网络服务模型和传输协议。 广义而言，互联网拥塞控制有两个作用：避免拥塞崩溃，并确保公平。 NDP实现了两者都没有一个明确的窗口适应机制

避免拥塞崩溃。 正如我们在第2.3节中看到的那样，NDP交换机通过确保大部分内容可以避免CP的拥塞崩溃

数据包使用链接。 如果数据包在接收器处被丢弃，也可能发生崩溃，就像Jacobson重传之前一样超时[26]或碎片[27,33]。 由于数据包微调，NDP发送者很少需要依赖RTO，因此崩溃不必要的重传是不可能的。

如果大部分数据包被丢弃靠近接收器，并且已经在其路径上早些移动了其他数据包，则可能发生崩溃[15]。 然而，在具有NDP的每包多路径路由的Clos拓扑中，在到核心交换机的上行链路上几乎不会修剪数据包，因为不可能将通信集中在那里。 当它们在上行链路上进行调整时，这是由于不完美的负载平衡造成的; 这是NDP的基于源的负载平衡提供的地方赢得交换机执行的每个数据包的随机ECMP。 即使在高负载情况下，这里修剪的数据包只占整个通信量的一小部分，例如，模拟运行全排列交通矩阵的128节点FatTree，其中每个节点从一个节点接收并以其链路速率发送到另一个节点 10Gb / s，我们看到当源负载平衡时，上行链路上的数据包调整率为0.01％，而交换机负载平衡时为2.4％。

重要的修整只发生在百万年前，大多数数据包在从架顶式交换机到主机的链路上被修剪，而少数则在上部交换机和下部交换机之间进行修剪。 因此，由ToR交换机修剪的数据包在拓扑结构中早些时候只有很少的位移数据包。 公平。 NDP在不需要额外机制的情况下实现了极好的公平性。 所有竞争的流程都从同一个窗口开始，所以不必担心会聚。 流量竞争的主要原因是接收器的容量，接收器对发生的事情有完整的看法。 通过对属于不同连接的拉队列中的数据包使用公平排队机制来实现接收者公平性。 最后，故意的不公平是可能的，因为接收者知道自己的优先级，并且可以比低优先级流量更经常地提取高优先级流量

一种不能被接收机管理的不公平现象是一个接收机流向同一个ToR交换机上的另一个接收机的情况。 新民主党在一个RTT中缓解了这种不公平现象因为在此之后，PULL的接收器起搏消除了过载。

NDP的局限性

我们在§5和§6中的实验评估表明，即使在不对称的情况下，无论流量模式如何，NDP在完全配置的折叠Clos拓扑中都非常接近最优。 这里我们讨论NDP在这些网络之外的局限性

在BCube [19]和Jellyfish [36]等非对称拓扑结构中，NDP的表现会很差，因为它会将数据包喷射到不同的这些路径在网络负载很重时使用成本很高。对于这样的网络，基于发送者的每路径多路径拥塞控制已经显示出效果很好[31]; 它仍然是一个开放的问题如何使我们的基于拉的接收机驱动协议协调每路拥塞控制。

对于核心持续拥塞的大量超额认购网络（如NDP），拥塞控制也是可取的即使在第一次RTT之后，积极的设计也会导致连续的数据包修整。 我们在评估中表明，在这种情况下，NDP仍然比DCTCP提供更好的性能（请参阅第6.3节），但是某种形式的拥塞控制对减少服务器重传负载很有用。

最后一个问题是关于部署：当P4交换机广泛部署在数据中心时，运行NDP与部署交换机实现（第4节）和终端系统堆栈一样简单。 但是，NDP可能会阻止竞争的TCP流量。 但是，通过从不同队列提供NDP和TCP，以及它们之间的公平排队来确保与TCP共存。 TCP队列将会更大（100个数据包），而NDP将会很小（8个数据包），再加上一个类似大小的头部队列。

4 履行

我们在Linux终端系统中实施了NDP，基于DPDK的软件开关[13]，使用10Gb / s NetFPGA的硬件开关SUME [41]平台和P4 [29]。我们还在htsim高速网络模拟器中实施了NDP，基于[31]的数据中心网络实施。 我们使用Linux和NetFPGA实现在实际硬件上小规模演示性能，并使用仿真来演示NDP的缩放属性。我们还开发了NDP交换机的P4实现，作为概念证明，NDP处理非常简单，可以轻松部署在可编程交换机中

Linux实施。我们的Linux NDP实现的目标是调查NDP性能并验证NDP协议设计。通常我们会希望NDP能够在OS内核中实现，以便准确地控制时序。为了允许快速实验，我们的方法改为在用户空间中实现NDP，使用DPDK库实现低延迟网络访问，并使用专用内核来确保准确的PULL调速和低延迟重传。该体系结构如图5所示。NDP核心进程调解NIC访问并维护拉队列因为所有的NDP连接都必须共享这个。核心进程还处理由NACK引起的快速重传。库为应用程序提供NDP API，并通过共享内存与核心进程交互，传递诸如connect和listen via之类的命令一个通讯环形缓冲区，以及每个活动套接字通道的数据三个环形缓冲区：RX，TX和RTX，以及一个共享缓冲池。该库还处理由于超时而导致的数据包重传

NDP核心主循环检查新的应用程序注册，运行来自库实例的命令，从新连接的TX环发送数据包的第一个RTT，并处理传入数据包。 到达数据时，ACK和NACK数据包被放置在适当的套接字的RX环中。 到达PULL将导致数据包从中发送套接字的RTX或TX环，RTX优先。到达数据数据包（或标题）会将PULL添加到拉取队列中

一个单独的PULL队列线程运行在其自己的CPU内核上，在适当的时间一个接一个地出队这些PULL，并发送他们; 目前该线程进行了调整以确保适当的粒度，但是将来可能会通过NIC支持来避免这种开销。

如果设置了SYN位并且先前调用了监听，数据包也可以触发新套接字的初始化。 NACK是传递给库以避免虚假超时，但NDP核心还将相应的缓冲区索引添加到套接字的RTX环中，允许非常快速的重传。

由于NDP是零RTT协议，库中的连接命令仅通知NDP内核有关新的活动套接字。数据发送时连接将建立。listen命令通知NDP核心一个新的被动套接字，但它也为任何传入连接保留了一些套接字，因为缺少初始握手意味着我们必须准备好接收传入的数据包序列。

支持NDP的硬件交换机。理想情况下，NDP的调整和优先级排队将在交换机ASIC中实现。我们使用NetFPGA-SUME平台[41]对这种解决方案进行了原型设计[41]，这是一个可重新配置的硬件平台，具有四个10Gb / s以太网接口，将Xilinx Virtex-7 FPGA与QDRII +和DDR3存储器资源。

图6显示了高层次的NDP交换机设计。数据包通过10Gb / s接口之一进入，并存储在36Kbit接口输入队列。 仲裁器使用不完全轮询（DRR）调度策略从输入队列获取数据包，并通过256位宽200MHz总线将它们提供给L2交换逻辑，速度足够快支持超过40Gb / s。 许多小包到达时DRR确保一个界面和许多大界面到达其他界面接收小包的输入队列不会溢出，在传统的L2转发决策之后，数据包达到NDP逻辑。 每个输出端口具有低优先级和高优先级输出队列，每个长度为12KByte。NDP控制数据包被转发到高优先级队列。 对于剩余的分组，NDP逻辑检查低优先级队列的长度，使队列队列队列中的队列长度排列数据包是否有空间。 否则，数据包将被修剪并放置在高优先级队列中。如果该队列已满，则数据包将被丢弃。请注意，完整的实现应该随机决定是否修剪数据包队列中的最后一个数据包或当前数据包分解相位效应。

NDP交换机使用63561个LUT（使用Virtex7容量的14.6％），77176个FlipFlops（8.9％）和231个RAM块（15.7％）。 相比之下，参考L2交换机使用11.4％，8.1％和13.2％因此NDP增加的复杂度很小。 在其他资源中，80％用于优先输出队列。

NDP交换机在P4中的实施。 如图7所示，该设计假定入口和出口管线之间至少存在两个队列，egress\_priority元数据决定哪个数据包进入哪个队列。NDP修改在简单的开关设备上演示，假设有一个单一的输出接口，但它们可以添加到任何P4开关，并且很容易修改为处理多个输出端口。

实现需要知道两个每端口队列的大小，以决定是否应该修剪数据包。为此，它可以利用当前队列大小寄存器来决定是否将数据包发送到优先队列;然而，并非所有的P4平台都有这个寄存器，所以我们选择通过计数所有数据包来实现具有类似功能的寄存器进入正常的缓冲区和进入出口管道的数据包。由于P4中的匹配/操作表只匹配分组数据，我们使用额外的表（Readregister）从寄存器读取qs并将其保存为分组元数据。如果qs低于允许的缓冲区大小，数据包将进入正常队列。一旦我们达到了门槛，数据包将会被截断（使用称为truncate的P4原语操作）并馈入优先级队列。没有数据有效载荷的NDP分组由于Directprio表，自动进入优先级队列。出口管道只处理队列大小簿记：qs如果数据包来自正常队列，寄存器会减少。

5 进化

我们希望了解NDP如何在具有实际工作量的大型数据中心中执行，但目前缺乏运行此类测试的能力。相反，我们使用NetFPGA NDP交换机来描述我们的Linux NDP实施，以了解小规模性能以及NDP与主机操作系统的交互方式。 接下来，我们将Linux实现与我们的模拟器进行比较，以确定其程度真实世界的文物会影响模拟的真实性。 最后，我们在模拟中评估NDP，以研究它如何扩展。我们使用各种流量模式和场景，并将其行为与交换机中使用ECN的MPTCP，DCTCP以及使用无损以太网的DCQCN [40]进行比较。

我们还使用参考P4开关对我们的P4 NDP开关进行了测试，以验证其正确性; 由于参考P4软件开关性能差，我们省略了这些运行的结果。

5.1LINUX NDP表现

NDP旨在提供低延迟传输。为了评估我们的Linux实施的延迟而没有混淆因素，我们连接了两台服务器，并运行了一个应用程序重复的RPC调用，发送和接收1KB的数据进行测量应用程序级别的延迟。我们比较我们的NDP原型和Linux内核TCP和TCP Fast Open（TFO），这是一种由Google提出的TCP优化，允许在TCP SYN上发送数据。 注意TFO不保证连接只处理一次，然而 NDP 这么做了。

如图8所示，使用NDP的中等延迟时间为62μs。TCP

快速打开需要四倍的时间，常规TCP需要五次更长的时间。 NDP RPC延迟与报告的RDMA延迟相当在最近的作品。

NDP如何实现如此低的延迟？ 为了理解，我们还通过DPDK实现了一个简单的ping应用程序，并进行了测量

延迟1KB ping。使用DPDK发送ping只需22μs，并获得答复。这意味着NDP协议处理并且应用程序处理（40μs）主导NDP RPC时间。NDP依靠DPDK，将一个核心专门用于数据包处理网卡缓冲区直接映射到用户空间。相比之下，TCP使用中断并在内核和用户空间之间复制数据。

为了理解NDP取得的成果，我们最初考察了中断和数据包拷贝的成本，但是这些开销都在最多50μs，所以不要解释我们观察到的巨大差异，尤其是与TFO相比。 经过进一步检查后，我们发现CPU的深度睡眠状态是导致差异最大的原因：由于TFO和TCP都依赖中断，CPU进入深度睡眠状态，唤醒大约需要160μs，严重影响延迟。我们禁止比C1更深的睡眠状态，并绘制图8中TFO和TCP的延迟。TFO和TCP现在做得更好，但NDP的延迟仍然刚刚超过TFO的一半，而TCP的延迟只有三分之一。原则上，如果TFO轮询数据而不是使用中断，TFO可以进一步优化以获得与NDP类似的性能。

聚播。传输协议最困难的流量模式低延迟的处理是满足的。我们将评估小型浓缩酒使用我们的Linux堆栈和NetFPGA交换机，以及更大的模拟器。我们的测试平台是一个8服务器的双层FatTree拓扑使用六个四端口NetFPGA NDP交换机构建。

作为我们的第一个实验，我们运行了7比1的incast：一台服务器上的前端应用程序向其他七台服务器发出同时请求，并立即回复。流量将集中在拓扑中的许多交换机上，可能会导致更远程服务器的不公平。 我们改变聚合反应的大小，并测量最后一次流动的完成时间。图9显示了响应规模在10KB和1MB之间的TCP和NDP的中位数和第90百分位完成时间。NDP在理论最佳完成时间的5％以内，NDP的90％百分位数在中位数的10％以内; 这两条线在图中重叠。

TCP的中值流量完成时间也随着响应大小线性增长，但NDP速度快四倍。TCP的中值流量不会遭受超时，使用快速重传从丢失中恢复。还有其他几个原因，包括三次握手，中断延迟，堆栈处理，附加数据副本，进程调度和次优拥塞响应等。这些加起来会伤害响应时间。 TCP的第90百分位是占主导地位的通过重传超时，MinRTO在Linux中为200ms。降低MinRTO并禁用延迟的ACK可能会降低这种影响以避免它们触发虚假重传。

优先级的好处。NDP的基于拉式的设计可以很好地控制流程完成时间，特别是在目标链路是瓶颈的常见情况下; 这将NDP分开从大多数现有的解决方案中，传输是围绕基于发送者的拥塞控制而建立的。

我们检查一个主机收到一个发件人的短流量和另外六个发送者的长时间流量的情况; 所有的流程都是同时开始的，所以特别是对于第一个往返时间的最后一跳来说，会有很大的争用。默认情况下，NDP会将所有发件人的速度调整为链接的1/7。然而，在这种情况下，接收机通过在较长的流水线之前发送它的拉力来优先考虑短流量。

图10显示了短流量发送200KB时的结果：当没有竞争流量时（标记为“闲置”），我们测量流量完成时间（FCT），并在与其他六个发送方竞争时与FCT比较，有和没有优先次序。 优先排序工作非常好：FCT的短当使用优先级时，流量与长时间流量竞争仅增加50μs，而当不使用优先级时，流量增加500μs。我们测试了流量大小从10KB到1MB不等：在所有情况下，空闲和优先级之间的差异小于50μs。第一次RTT之后，期间来自长流的数十个数据包也被传送，短流程完全占用接收器的链路直到完成。

6 模拟

在研究大规模实验之前，我们先研究模拟器的表现与我们的Linux实现在同一配置中的表现有何不同。我们确定的主要差异是主机处理延迟，以及PULL的速度。我们的模拟器完全匹配PULL的速度，但我们的实现却不能。

为了理解处理延迟的影响，我们将两台服务器背对背连接并测量实现的吞吐量，作为初始窗口（IW）的一个功能。 图11显示为了达到相似的吞吐量，原型的初始窗口必须是25数据包而不是15个（即大于15KB）; 这些额外的数据包在终端系统中被缓冲，并且需要覆盖不由模拟器建模的主机处理延迟。这意味着源自模拟器的NDP延迟稍微过于乐观。 然而，TCP的未模拟处理延迟更高，所以任何来自模拟器上的比较略微偏向于支持TCP。

图12显示了1500B和9000B数据包的pull的实际间距，由发件人测量。而中位数值匹配目标间距（分别为1.2μs和7.2μs），与1500B数据包有些差异。

为了观察这种不完美的拉距的影响，我们向模拟器添加了代码，该模拟器从中拉出间隔距离通过实验测量分布。 首先，我们重新执行上面图11中的成对转移; 标记为“发件人CDF”覆盖的结果“完美”曲线：pull间距不会影响吞吐量，因为窗口足够大以覆盖PULL中的小“间隙”。

接下来，我们运行了一个组合实验，其中有432节点的FatTree中的每个节点发送到一个节点并从另一个节点接收，

完全加载数据中心。我们使用了1500B的数据包，并将完美的pull间距与我们的实验结果中的数据进行了比较。吞吐量的差异是1.2％，可以忽略不计

最后，我们进行了一次200：1的incast试验，试验了不同的流量大小，并测量了最后一次流量的完成时间。 图13显示，当使用不完美的pull间距时，流的完成时间没有明显的差异。这些实验一起表明真实世界的人工产品对NDP的影响小，并且对于基于大规模模拟结果的预测提供信心。

6.1 与现有工作的比较

我们通过利用Clos数据中心网络运行一个排列来衡量NDP的能力：这是一个最糟糕的流量模型，其中每个服务器都是打开一个单独的长期运行的连接到另一个随机的服务器，以便每台服务器都确保有一个入站连接。我们将NDP与MPTCP [31]，DCTCP和新提出的DCQCN协议进行比较，DCQCN协议本质上是一种运行DCTCP在无损以太网上的方式。NDP交换机使用8个数据包输出队列，而为了确保良好的性能，DCTCP和MPTCP使用200个数据包输出队列，DCQCN使用200个缓冲区/每个端口，在接口之间共享。 DCTCP和DCQCN标记建议的阈值分别是30和20个数据包。

图14显示了每台主机按升序顺序达到的吞吐量。DCTCP和DCQCN使用单一路径并受ECMP产生碰撞的影响：平均利用率约为40％尽管如此，并且，仍有一些流量低于1Gb / s，尽管有充足的容量来提供每个流10Gb/s.MPTCP效果更好：利用率为89％，最差流量达到6Gb/s。NDP有92％的利用率，并提供很多公平性：即使最慢的流量也能达到9Gb / s。

缓冲区大小对小流量完成时间有什么影响？ 我们预计NDP应该受益于较小的网内缓冲区，因为DCTCP和DCQCN具有较长的流程完成时间。两个节点重复交换90KB传输文件以测试延迟，而所有其他节点都会每个源向一个随机目标发送四个长运行连接。由于在90KB流量的源和目的地没有争议，这测试了网络中站队列对其他空闲主机运行的短流量的影响。

短流程完成时间的CDF如图15所示.NDP的最坏情况延迟仅为理论最佳传输时间的两倍在空闲网络中，与DCTCP相比，中位数低三倍，在99％时低四倍。 主要原因这种差异在于NDP的缓冲区远远小于DCTCP在超负荷Fat Tree网络中实现的缓冲区。DCQCN具有比DCTCP稍差的性能，因为PAUSE帧被偶尔触发。MPTCP实现了所有解决方案中最差的FCT，由于其贪婪的网络缓冲区，中位数和尾数比NDP大10倍。 在这些实验中，NDP和MPTCP实现了80％的网络利用率，DCTCP和DCQCN达到〜75％。接下来，我们测试一个聚集流量模式，前端将工作散发给许多后端服务器，然后接收他们的回复; 这样用例在网络搜索应用中很常见，等等。 同时到达的响应对通往前端的交换机端口产生巨大的缓冲压力，导致同步丢失。 我们改变后端服务器的数量，同时保持响应大小恒定（450KB）并测量流程完成时间。一般来说，最后流量完成时间是所关注的度量，但在图16中，我们还显示了最快流量的完成时间，以突出显示不同方案的“公平性”。

即使我们使用Vasudevan等人的积极的小型定时器[38]，MPTCP（和任何尾部丢失TCP变体）也被同步化导致大型和不可预测的FCT的损失。DCTCP使用ECN获得早期反馈，以及大型共享缓冲区开关来吸收初始突发，因此它比传统的TCP尾部切换更好。DCTCP的平均速度比理论最优慢5％。 DCQCN和NDP做得更好，完成时间仅比最佳时间慢1％。

接下来，请注意不同协议的传播完成时间。DCTCP在其最快和最慢的流量之间有很宽的范围（高达七倍）。新民主党有一个非常平衡的分配，最慢的流量最多花20％的时间完成比最快的一个; 这是NDP交换机的直接后果。最后，DCQCN的分配非常紧张，直到350KB响应大小，当它在ECN标记制度下运行时（门槛比DCTCP小）。除此之外，无损操作正在狠狠地发生扭曲流程完成时间。

我们还为一个单播发送者启用了NDP优先级：它的拖拽将被放置在接收者的拖拽队列的头部。 优先顺序是非常有效的：首选流程的完成时间仅为1ms，100个发送者为3.5ms，432个发送者为432ms

6.1.1 INCAST流量的负面效应

incast如何影响附近的交通？我们进行了两个独立的实验，每个实验都涉及一个大的incast 在第一个实验中，incast是长期存在的并且与排列流量矩阵一起运行。 这里感兴趣的度量是网络总利用率：NDP达到92％和DCTCP的40％利用率，与单独运行的排列相同。但是，使用DCQCN时，网络会出现拥塞崩溃现象：利用率下降到17％，因为incast触发PFC，严重影响数据中心大多数流的吞吐量

在我们的第二个实验中，如图18所示，我们对主机1运行一个长寿命流，然后启动一个短暂的64对1的incast流量模式主机2，每个流量发送900KB。两个主机都在相同的ToR交换机。结果如图19所示。对于DCTCP，在ToR交换机和汇聚处，incast都会导致损失交换机端口通向ToR。 长流和内流流动需要一些时间来恢复。 在t = 0.17时突发超过10Gb / s当重传到达时出现，允许已经收到的数据最终被发布到应用程序。

使用DCQCN时，可以防止损失，并且可以快速完成incast流动（注意不同的x轴），但是PFC会导致上游开关重复暂停，从而影响长流。 由于暂停造成的这种附带损害是PFC的主要缺点。

使用NDP，incast会在第一个RTT期间导致修剪。由于这个原因，长流量的吞吐量小于1ms初爆。 在第一次RTT之后，接收器对剩余的incast分组进行步进，并且长流量恢复以再次获得完整吞吐量

6.2 敏感性分析

NDP使用非常小的开关缓冲器和固定的初始窗口（IW）在发件人; 这些是NDP中唯一的两个参数网络。 IW整合了有关瓶颈速度和网络延迟的信息，并由管理员设置。我们使用模拟来了解这些参数如何影响性能，以及良好的性能是否需要仔细调整

这里最重要的指标是最差情况下的流量矩阵，即置换。 我们改变IW的值，并测量每个主机针对各种交换缓冲区大小实现的平均吞吐量; 我们将结果绘制在图17中。首先，注意需要20个数据包的IW才能充分利用网络; 此外，当IW小于15时，交换缓冲区的大小就会变大没关系。无论IW的价值如何，使用六个数据包缓冲区都会略微低估网络利用率（90％的利用率）。 略大的队列（8个数据包）导致95％以上的使用率; 这是我们在本文中使用的队列大小。最后请注意如何进一步增加IW减少吞吐量：这是因为创建更多的缓冲压力会导致更多的标题。当使用1.5KB数据包时，30个数据包的IW导致95％网络利用率。考虑到总尺寸，结果是显着的以字节为单位的交换缓冲区仅为12KB。

对于incast工作负载，较小的IW更适合于较少的数据包在第一个RTT中被修剪。我们在模拟中使用了30的IW并在部署中，除非另有说明。

更大的拓扑结构。 尽管我们的数据包级仿真器速度很快，但仍然需要花费数十分钟来模拟432节点FatTree中的置换矩阵，这是我们在整个评估过程中使用的拓扑。 先前的工作已经表明，当100个分组缓冲器联合使用时，置换流量矩阵对于128,1024和8192节点的胖树[31]具有类似的宏观行为与多路径TCP。八个分组交换机缓冲区是否足以保证在大型网络中也能高度利用NDP？

我们使用8个数据包缓冲区，9KB MTU和30个IW，运行了更大拓扑的排列实验。在8192节点FatTree中，网络利用率从98％（128节点FatTree）缓慢降至90％。与使用八个子流（[31]，图2）的MPTCP相比，NDP的利用率提高了5％，而使用的缓冲区不到其十分之一。

大规模种植NDP可以应对的种植大小是否有任何根本的限制？我们在一个8192节点的FatTree中运行从一个流（不包含incast）到8000个同时流，每个270,000个字节（30个数据包）的大量流量，并测量流量最后流量的完成时间。图20a显示了时间开销占最佳理论上溢时间的百分比;这假定到接收器的链路完全饱和，直到最后一个流完成，并且每个分组只接收一次。同一个23个数据包IW（图11所示为非传输流量的良好默认值），小型百事可乐看到最差的开销，但仍然完成在2％以内的最佳。这些开销是由于标题转发在第一次RTT中，由于没有完全填充接收器的链接最终的RTT。对于更大的传播，时间开销可以忽略不计。尽管这些开销很低，但如果它们能够进一步降低应用程序知道一个大的incast可能。初始窗口曲线显示了十个数据包和一个数据包。十个数据包接近这是可以通过卸载网络填充管道的最小窗口和存储转发交换机。对于小于八流的inc inc，使用一个数据包IW的开销很高，因为需要在那里每个RTT发送至少八个数据包以填充接收器的链路。

虽然我们最关心延迟，但检查成本发送者为实现这种低延迟付出的代价也很有趣。图20b显示每个数据包重传的平均次数，以及通知发件人需要重发的机制（返回发件人跳回或NACK）。 对于较小的浓度，NACK是主要机制。超过100流，返回发件人成为主要机制。在2000流量以上，有些数据包在通过之前遭遇第二次返回发送者。 即使是最大的传播和23分组的IW，平均重传次数也不会超过1次。然而，对于那些知道他们将创建大型incast来减少初始窗口的应用程序来说，这是有意义的。

发件人限制流量。考虑图21中的流量模式，其中主机A发送给主机B，C，D和E，并且主机F也发送到主机E.在正常情况下，到E的通信量将在源A和F之间平均分配，但在这种情况下，A不能发送足够的E的链接。从E中拉出来的东西会被浪费掉吗？

我们模拟了这种情况。吞吐量在图21中列出。从A到E的链路都是饱和的，并且由A产生的四个流量几乎完美地分割了容量。新民主党如何取得如此好的结果？ 原因是E执行在其拉队列上公平排队。在E处，总是有PULL数据包排队等候F.但是，由于来自A的数据包不常到达，所以这个公平队列确保在到达的数据包能够生成它时立即发送用于A的PULL包。这确保E以相同的速率向A发送拉包。A向E发送数据包.E的其余部分拉到F，确保E的链路保持满。B，C和D不能比A发送PULL更快的数据包到达。 因此，PULL从B，C，D和E大致同等地到达A，并且这个确保A的传出链接既充分又平等。

谁需要封包修整？一个有效的问题是我们能否在不改变网络交换机的情况下获得NDP的好处。PHost[16]是一个接收器驱动的传输，运行非常小的数据包缓冲区和每个数据包的ECMP，但不使用数据包修整。我们通过改变htsim中的TCP来实施pHost; pHost发送器在第一个RTT中以线速度突发，以获得最佳短流性能。为了理解pHost的优点和修剪的附加价值，我们在8个数据包缓冲区的定期拖尾432节点FatTree网络上运行pHost。图16中的432对1 incast将pHost 1s至1.5s完成。这不仅比NDP（140ms）差十倍，而且比MPTCP差4-5倍。我们还运行了排列流量矩阵，发现pHost尽管其使用率仅达到70％使用包喷雾（NDP达到95％的利用率）

处理不对称。到目前为止我们测试的所有网络都是对称的; 在这种情况下每包ECMP工作井。这是更难的当网络不对称时，例如由于失败而做得很好。我们在一个128节点的拓扑中运行了一个置换实验核心和上层交换机之间的一条链路速度降低到1Gbps。图22中的结果显示NDP和MPTCP很好地处理故障，因为它们保留了每个路径的拥塞信息并将流量从拥塞的路径中分离出来。 路径惩罚机制NDP使用（见第3.2.3节）在非对称拓扑中至关重要：如果没有它，NDP的效果很差，15流量到达只有3Gbps的吞吐量。一些DCTCP流程也受到影响：最严重的连接只能达到0.4Gbps。

超载。在完全配置的Clos拓扑上，NDP的行为接近最佳。但是，并非所有数据中心都提供完整的横截面带宽; 当核心网络超额订阅或者数据包大小小于MTU时会发生什么？ 在这种情况下，我们预计NDP将修剪许多数据包，并在修剪发生时实现较低的压缩比

我们使用来自Facebook网络的测量来设置我们的实验[34]。 模拟的拓扑结构是一个三层FatTree包含512个通过10Gbps链路连接到ToR交换机的服务器。从ToR到汇聚交换机的上行链路连接是四个次数低于服务器连接，给予4：1超额认购。

在[34]中介绍的三种网络流量（网页，缓存和Hadoop）中，我们使用最不利于NDP的方式：网络流量模式具有非常小的数据包，压缩性差，几乎没有机架级别的局部性， 几乎所有的交通必须遍历超额认购的核心。 流量大小来自于

分布[34]（图6a）和流动到达是闭环，中值流动间隙为1ms。 我们通过增加每个主机的同时连接数量来改变负载，并通过检查核心利用率来衡量负载。

我们比较了NDP和DCTCP，并绘制了图23中的流程完成时间分布图。我们运行两个独立的测试：首先，我们运行一个适度加载的网络（每台主机同时连接五个），其中NDP主机发送的所有数据包的40％通过ToR交换机。 在这种情况下，新民主党的中位数FCT是DCTCP的一半，在第99百分位是第三位。

接下来，我们更加加载网络，每台主机同时向其他随机选择的主机提供10个连接。70％的数据包在第一个ToR交换机上被修剪; 这与考虑到核心网络超额预订为4：1的最坏情况非常接近。尽管如此，NDP性能强劲，在中值和尾部方面都比DCTCP提供稍好的性能。NDP不受影响来自拥挤崩溃; 我们观察到，如果一个数据包超过了ToR交换机，它很可能会以一定的概率到达目的地被修剪只有2-5％。

总而言之，即使在这个超额认购的网络中，NDP也很健壮，性能比DCTCP好。这并不是说NDP应该像在大规模拥塞的网络中一样使用：在这种情况下修剪的数据包数量是浪费的。 当大多数数据包被修剪时，一个简单的拥塞控制算法可以降低提取速率，避免持久的过载，但不需要保守，因为NDP工作良好，没有拥塞控制。

7相关工作

有大量工作旨在解决数据中心传输的各个方面：大多数重点在于实现低延迟[5，6,16,22,28,30]或高通量[2,12,31]。 我们已经详细讨论过，并且与以前最相关的部署工作进行了比较，即低延迟的DCTCP / DCQCN和高吞吐量的MPTCP; 这里我们概述其余部分。

pFabric [6]旨在通过基于严格优先级的数据包交换来实现数据中心规模的最短时间流量调度。尽管理论上吸引人，但pFabric很难部署，因为它委托主机正确优先处理流量。 赫尔[5]保留能力以确保低短流量等待时间，并使用幻像队列来识别潜在的拥塞; 它需要在端点进行精确的数据包调步。 Fastpass [30]使用集中式调度来实现低延迟，但规模有限。 如果可以修改应用程序以明确规定最终期限，那么有很多提案将有助于包括PDQ，截止时间感知数据中心TCP（D2TCP）和D3 [22,37,39]。

TIMELY [28]，一种针对无损网络的DCQCN的替代拥塞控制机制，完全依赖于RTT作为拥塞度量。与DCQCN一样，TIMELY不能完全防止暂停帧及其对无辜网络流量的负面影响。总的来说，所有这些工作的最大局限性是它们只关注低延迟，而忽略网络利用率或大流量性能。

另一方面，研究人员一直热衷于通过集中调度来解决因流量过大而导致的冲突在Hedera [2]中，通过分组喷洒协议的变化[12]和Presto [21]，或者通过Conga网络子流处理[3]和开关重新设计LocalFlow [35]。这些作品不是针对短片流动完成时间，并获得相当于FCT的FCT未经优化的网络运行大型数据包缓冲区。

NDP脱颖而出，因为它在所有通信条件下都实现了低延迟和高吞吐量。

8 总结

我们提出了NDP，一种新的数据中心网络架构，包括修改的交换机排队算法，每个数据包的多路径转发，以及利用这些网络机制的新型传输协议。NDP展品出色的低延迟行为，无论是在我们的实施还是在大规模模拟中。 它比不依赖无损以太网实现低延迟的DCQCN机制在不同的工作负载之间提供了更好的隔离。

由于需要，我们目前的实施在终端系统所需的CPU资源方面是相当昂贵的精确调度PULL和低延迟重传。 这两种功能对于以太网NIC来说都非常简单; 事实上，廉价的WiFi网卡已经可以处理重传和小心的数据包时序。我们预计NDP将被大规模部署智能NIC将大大降低NDP的CPU开销。

致谢

这项工作部分由SSICLOPS H2020项目（644866）资助。 我们要感谢匿名审稿人Giuseppe Lettieri和我们牧羊人Amin Vahdat的非常有用的反馈，这有助于我们改进本文。