RDMA over Commodity Ethernet at Scale(Introduction)

> With the rapid growth of online services and cloud computing, large-scale data centers (DCs) are being built around the world. High speed, scalable data center networks (DCNs) [1, 3, 19, 31] are needed to connect the servers in a DC. DCNs are built from commodity Ethernet switches and network interface cards (NICs). A state-of-the-art DCN must support several Gb/s or higher throughput between any two servers in a DC. TCP/IP is still the dominant transport/network stack in today's data center networks. However, it is increasingly clear that the traditional TCP/IP stack cannot meet the demands of the new generation of DC workloads [4, 9, 16, 40], for two reasons. First, the CPU overhead of handling packets in the OS kernel remains high, despite enabling numerous hardware and software optimizations such as checksum offloading, large segment offload (LSO), receive side scaling (RSS) and interrupt moderation. Measurements in our data centers show that sending at 40Gb/s using 8 TCP connections chews up 6% aggregate CPU time on a 32 core Intel Xeon E5-2690 Windows 2012R2 server. Receiving at 40Gb/s using 8 connections requires 12% aggregate CPU time. This high CPU overhead is unacceptable in modern data centers. Second, many modern DC applications like Search are highly latency sensitive [7, 15, 41]. TCP, however, cannot provide the needed low latency even when the average traffic load is moderate, for two reasons. First, the kernel software introduces latency that can be as high as tens of milliseconds [21]. Second, packet drops due to congestion, while rare, are not entirely absent in our data centers. This occurs because data center traffic is inherently bursty. TCP must recover from the losses via timeouts or fast retransmissions, and in both cases, application latency takes a hit. In this paper we summarize our experience in deploying RoCEv2 (RDMA over Converged Ethernet v2) [5], an RDMA (Remote Direct Memory Access) technology [6], to address the above mentioned issues in Microsoft's data centers. RDMA is a method of accessing memory on a remote system without interrupting the processing of the CPU(s) on that system. RDMA is widely used in high performance computing with Infiniband [6] as the infrastructure. RoCEv2 supports RDMA over Ethernet instead of Infiniband. Unlike TCP, RDMA needs a lossless network; i.e. there must be no packet loss due to buffer overflow at the switches. RoCEv2 uses PFC (Priority-based Flow Control) [14] for this purpose. PFC prevents buffer overflow by pausing the upstream sending entity when buffer occupancy exceeds a specified threshold. While some problems with PFC such as head-of-the line blocking and potential for deadlock are well known [22, 33], we see several issues such as the RDMA transport livelock, the NIC PFC pause frame storm and the slowreceiver symptom in our deployment that have not been reported in the literature. Even the root cause of the deadlock problem we have encountered is quite different from the toy examples often discussed in the research literature [22, 33]. We also note that VLAN [32] tags are typically used to identify PFC-enabled traffic in mixed RDMA/TCP deployments. As we shall discuss, this solution does not scale for our environment. Thus, we introduce a notion of DSCP (Differentiated Services Code Point) based PFC to scale RDMA from layer-2 VLAN to layer3 IP. Our RDMA deployment has now been running smoothly for over one and half years, and it supports some of Microsoft's highly-reliable and latency-sensitive online services. Our experience shows that, by improving the design of RoCEv2, by addressing the various safety issues, and by building the needed management and monitoring capabilities, we can deploy RDMA safely in largescale data centers using commodity Ethernet.

> 随着在线服务和云计算的快速增长，大规模的数据中心（DC）正在世界各地建立。连接 DC 中的服务器需要高速、可扩展的数据中心网络 (DCN) [1,3,19,31]。 DCN 由商用以太网交换机和网络接口卡 (NIC) 构建而成。最先进的 DCN 必须支持 DC 中任意两台服务器之间的几 Gb/s 或更高的吞吐量。 TCP/IP 仍然是当今数据中心网络中占主导地位的传输/网络堆栈。然而，越来越明显的是，传统的 TCP/IP 堆栈无法满足新一代 DC 工作负载的需求 [4,9,16,40]，原因有两个。首先，尽管启用了许多硬件和软件优化，例如校验和卸载、大段卸载 (LSO)、接收侧扩展 (RSS) 和中断调节，但操作系统内核中处理数据包的 CPU 开销仍然很高。我们数据中心的测量结果表明，在 32 核 Intel Xeon E5-2690 Windows 2012R2 服务器上，使用 8 个 TCP 连接以 40Gb/s 的速度发送会占用 6% 的总 CPU 时间。使用 8 个连接以 40Gb/s 的速度接收需要 12% 的总 CPU 时间。这种高 CPU 开销在现代数据中心中是不可接受的。其次，许多现代 DC 应用程序（例如搜索）对延迟高度敏感 [7,15,41]。然而，即使平均流量负载适中，TCP 也无法提供所需的低延迟，原因有两个。首先，内核软件引入的延迟可能高达数十毫秒[21]。其次，由于拥塞导致的数据包丢失虽然很少见，但在我们的数据中心中并非完全不存在。发生这种情况是因为数据中心流量本质上是突发性的。 TCP 必须通过超时或快速重传来恢复损失，在这两种情况下，应用程序延迟都会受到影响。在本文中，我们总结了部署 RoCEv2（基于聚合以太网的 RDMA v2）[5]（一种 RDMA（远程直接内存访问）技术 [6]）的经验，以解决 Microsoft 数据中心中的上述问题。 RDMA 是一种访问远程系统上的内存而不中断该系统上的 CPU 处理的方法。 RDMA广泛应用于以Infiniband[6]为基础设施的高性能计算。 RoCEv2 支持以太网 RDMA，而不是 Infiniband 与 TCP 不同，RDMA 需要无损网络；即，不得因交换机的缓冲区溢出而导致数据包丢失。为此，RoCEv2 使用 PFC（基于优先级的流量控制）[14]。 PFC防止缓冲与TCP不同，RDMA需要无损网络；即，不得因交换机的缓冲区溢出而导致数据包丢失。为此，RoCEv2 使用 PFC（基于优先级的流量控制）[14]。当缓冲区占用率超过指定阈值时，PFC 通过暂停上游发送实体来防止缓冲区溢出。虽然 PFC 的一些问题（例如队头阻塞和潜在的死锁）是众所周知的 [22, 33]，但我们在我们的模型中看到了一些问题，例如 RDMA 传输活锁、NIC PFC 暂停帧风暴和缓慢接收器症状文献中未报道过的部署。甚至我们遇到的死锁问题的根本原因也与研究文献中经常讨论的玩具示例有很大不同[22, 33]。我们还注意到，VLAN [32] 标签通常用于识别混合 RDMA/TCP 部署中启用 PFC 的流量。正如我们将要讨论的，该解决方案无法适应我们的环境。因此，我们引入基于 PFC 的 DSCP（差分服务代码点）概念，将 RDMA 从第 2 层 VLAN 扩展到第 3 层 IP。我们的 RDMA 部署现已顺利运行了一年半多，并且支持 Microsoft 的一些高可靠且对延迟敏感的在线服务。我们的经验表明，通过改进 RoCEv2 的设计、解决各种安全问题以及构建所需的管理和监控功能，我们可以使用商用以太网在大型数据中心中安全地部署 RDMA。

large segment offload (LSO)

在计算机网络中，**大段卸载**（**L**arge **S**egment **O**ffload，简称**LSO**）是一种在高带宽网络中用于减少CPU使用率和增加发送吞吐量的技术，该技术通过网卡对过大的数据分段，而无需协议栈参与。该技术还有一些别称，当应用于TCP时被称为**TCP段卸载(TSO)**，也有些时候被称为**通用段卸载(GSO)**。

与之对应，大接收卸载是用于接收大段卸载的数据的一种技术

当一个系统需要通过网络发送一大段数据时，计算机需要将这段数据拆分为多个长度较短的数据，以便这些数据能够通过网络中所有的网络设备(例如路由器、交换机)，这个过程被称作分段。

通常，这个过程由计算机系统中的协议栈完成，而大段卸载技术将这一过程交给网卡处理，从而减少了CPU使用率。

例如，一个64KB的数据在发送前通常会被分为46个小段，每一段1448字节(这个值与MTU有关)。如果网卡支持大段卸载技术，计算机既可以将这64KB数据直接交给网卡，网卡会将其拆分为不大于1448的小段，并根据TCP/IP协议栈提供的模板为每个小段增加传输层、网络层以及数据链路层头部。许多2014年后新推出的网卡都具备了这种技术。

一些网卡在处理TCP数据时使用TSO技术，即每个小段都包含完整的TCP头部。而对于其他传输层协议(例如UDP)则使用IP分片。

[补充介绍]

MTU

最大传输单元（英语：**M**aximum **T**ransmission **U**nit，缩写**MTU**）是指数据链路层上面所能通过的最大数据包大小（以字节为单位）。最大传输单元这个参数通常与通信接口有关（网络卡、串口等）。

因特网协议允许IP分片，这样就可以将数据报包分成足够小的片段以通过那些最大传输单元小于该数据报原始大小的链路了。这一分片过程发生在IP层（OSI模型的第三层，即网络层），它使用的是将分组发送到链路上的网络接口的最大传输单元的值。原始分组的分片都被加上了标记，这样目的主机的IP层就能将分组重组成原始的数据报了。

在因特网协议中，一条因特网传输路径的“路径最大传输单元”被定义为从源地址到目的地址所经过“路径”上的所有IP的最大传输单元的最小值。或者从另外一个角度来看，就是无需进一步分片就能穿过这条“路径”的最大传输单元的最大值。

RSS（receive side scaling）

RSS使网络适配器能够在多核计算机的多处理器内核上分配内核模式网络处理负载。
The distribution of this processing makes it possible to support higher network traffic loads than would be possible if only a single core were to be used.这种分布使得支持高负载网络成为现实（如果还是单核处理器，则很难实现）
在Windows Server 2012中，RSS得到了增强，包括拥有超过64个处理器的计算机。
RSS通过将网络处理负载分散到许多处理器上，并主动负载平衡TCP终止的流量来实现这一点。

此外，Windows Server 2012中的RSS为非TCP流量（包括UDP单播、多播和IP转发流量）提供了自动负载平衡功能，改进了RSS诊断和管理功能，提高了非统一内存访问（NUMA）节点的可扩展性，并通过促进内核调度器和网络堆栈对齐来提高资源利用率和资源分区

RSS为低延迟场景提供了以下额外好处：

- **并行接收处理：**可以通过在多个CPU上同时生成中断和DPC来指示从单个网络接口卡接收数据包。
- **保留订单数据包传递：**从单个网络接口卡接收到特定流的数据包，以便传递给TCP/IP协议驱动程序。
- **动态负载平衡：**随着主机负载的变化，RSS会重新平衡处理器之间的网络处理负载。
- **缓存位置：**由于来自单个连接的数据包被映射到特定连接的特定处理器状态，因此驻留在处理器的缓存中，从而消除了缓存敲击并提高了性能。
- **发送侧缩放：**RSS散列值由TCP/IP协议传递到出口路径上每个数据包中的网络接口卡，从而允许在同一CPU上指示发送完成。这使得在发送端可以更好地缩放。
- **Toeplitz Hash：**默认生成的RSS签名在统计学上是安全的，这使得恶意远程主机更难迫使系统进入不平衡状态。

以下是RSS的一些特定应用优势：

- RSS使Windows Server 2012能够在拥有超过64个处理器的大型服务器上进行优化扩展。
- RSS允许多NUMA节点服务器，如Web服务器、文件服务器和运行其他工作负载的服务器，高效扩展。
- 对于DirectAccess，RSS允许通过主动负载平衡转发流量来提高可扩展性。
- RSS使金融应用程序，如股票代码工厂和使用UDP作为传输的算法交易系统，能够随着负载的增加而扩展。
- RSS为应用程序提供输入/输出控制，以与堆栈对齐。
- RSS与Windows负载平衡和故障转移解决方案兼容。
- RSS为管理员提供了通过标准管理界面和诊断工具部署、管理和诊断RSS功能的能力。
- RSS通过将网络堆栈与内核调度器对齐来优化扩展所有工作负载。RSS在做出处理器选择决策时使用内核调度器提供的处理器列表。

[补充介绍]

RSS（另一个RSS，really simple syndicate）

RSS（**RDF网站摘要或真正简单的联合**）2(https://en.wikipedia.org/wiki/RSS#cite_note-powers-2003-1-2)是一个[网络提要](https://en.wikipedia.org/wiki/Web_feed "网络提要")3(https://en.wikipedia.org/wiki/RSS#cite_note-Netsc99-3)==允许用户和应用程序==以[标准化](https://en.wikipedia.org/wiki/Standardization "标准")、计算机可读的格式访问网站的更新。订阅RSS提要可以让用户在单个新闻聚合器中跟踪许多不同的网站，该聚合器不断监控网站的新内容，无需用户手动检查它们。新闻聚合器（或“RSS阅读器”）可以内置到浏览器中，安装在台式计算机上，也可以安装在移动设备上。

网站通常使用RSS提要发布经常更新的信息，如博客条目、新闻头条、音频和视频系列剧集，或分发播客。RSS文档（称为“提要”、“网络提要”、4(https://en.wikipedia.org/wiki/RSS#cite_note-GuardWF-4)或“渠道”）包括完整或摘要文本，以及[元数据](https://en.wikipedia.org/wiki/Metadata "元数据")，如发布日期和作者姓名。RSS格式使用通用XML文件指定。

尽管RSS格式早在1999年3月就发生了变化，5(https://en.wikipedia.org/wiki/RSS#cite_note-Qstart-5)在2005年至2006年期间，RSS被广泛使用，并且图标由几个主要网络浏览器决定。[[6]](https://en.wikipedia.org/wiki/RSS#cite_note-6)RSS提要数据使用称为新闻聚合器的软件呈现给用户，内容的传递称为[网络辛迪加](https://en.wikipedia.org/wiki/Web_syndication "网络辛迪加")。用户通过在阅读器中输入提要的URI或单击浏览器的提要图标来订阅提要。RSS阅读器定期检查用户的提要以获取新信息，如果启用该功能，则可以自动下载

RDF （Resource Description Framework）

资源描述框架（**RDF**）是万维网联盟（W3C）标准，最初设计为元数据的数据模型。它已被用作描述和交换图形数据的一般方法。RDF提供各种语法符号和数据序列化格式，其中Turtle（Terse RDF三重语言）是目前使用最广泛的符号。

RDF是由三重语句组成的有向图。RDF图语句表示为：1）主体的节点，2）从主体到谓词对象的弧，以及3）对象的节点。语句的三个部分中的每一个都可以通过统一资源标识符（URI）来识别。对象也可以是字面值。这个简单、灵活的数据模型具有很大的表达能力，可以代表复杂的情况、关系和其他感兴趣的事物，同时也具有适当的抽象性

RDF数据模型1(https://en.wikipedia.org/wiki/Resource_Description_Framework#cite_note-1)类似于经典概念建模方法（如[实体关系](https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model "实体-关系模型")或类图）。它基于以subject-predicate-object（称为*三元组*）的形式表达方式对资源（特别是网络资源）进行陈述的想法。主语表示资源，predicate表示资源的特征或方面，并表示主语和object之间的关系。

例如，在RDF中表示“天空有蓝色”概念的一种方式是三重：表示“天空”的主题，表示“有颜色”的谓词，表示“蓝色”的宾语。因此，RDF使用subject而不是object（或entity），与面向对象设计中实体-属性-值模型的典型方法相反：实体（天空）、属性（颜色）和值（蓝色）。

RDF是一个具有多种序列化格式的抽象模型（本质上是专门的文件格式）。此外，资源或三元组的特定编码可能因格式而异。

interrupt moderation(中断审核)

To reduce the number of interrupts, many NICs use *interrupt moderation*. With interrupt moderation, the NIC hardware will not generate an interrupt immediately after it receives a packet. ==Instead, the hardware waits for more packets to arrive, or for a time-out to expire, before generating an interrupt. ==The hardware vendor specifies the maximum number of packets, time-out interval, or other interrupt moderation algorithm.

The measured round-trip time for a packet is one of the most commonly used techniques to determine the network bandwidth between two endpoints. However, when interrupt moderation is enabled, receiving a packet does not generate an immediate interrupt and therefore the perceived round-trip time for a particular packet becomes larger than the average time. To allow accurate measurement of round trip time for a packet, NDIS provides the ability to disable and enable interrupt moderation on demand.

kernel software

"Kernel (computer science)" redirects here. For other uses, see Kernel (disambiguation) § Computing.

The **kernel** is a computer program at the core of a computer's operating system and generally has complete control over everything in the system. Kernel is also responsible for preventing and mitigating conflicts between different processes1(https://en.wikipedia.org/wiki/Kernel_(operating_system)#cite_note-Linfo-1) It is the portion of the operating system code that is always resident in memory2(https://en.wikipedia.org/wiki/Kernel_(operating_system)#cite_note-2) and facilitates interactions between hardware and software components. A full kernel controls all hardware resources (e.g. I/O, memory, cryptography) via device drivers, arbitrates conflicts between processes concerning such resources, and optimizes the utilization of common resources e.g. CPU & cache usage, file systems, and network sockets. On most systems, the kernel is one of the first programs loaded on startup(after the bootloader). It handles the rest of startup as well as memory, peripherals, and input/output (I/O) requests from software, translating them into data-processinginstructions for the central processing unit.

有不同的内核架构设计。整体内核完全在单个地址空间中运行，CPU在主管模式下运行，主要是为了速度。微内核在用户空间中运行其大多数但不是所有服务，3(https://en.wikipedia.org/wiki/Kernel_(operating_system)#cite_note-3)像用户流程一样，主要是为了弹性和[模块化]

(https://en.wikipedia.org/wiki/Modular_programming "模块化编程")。4(https://en.wikipedia.org/wiki/Kernel_(operating_system)#cite_note-mono-micro-4) MINIX 3是微内核设计的一个显著例子。相反，Linux内核是单一的，尽管它也是模块化的，因为它可以在运行时插入和删除可加载的内核模块。

计算机系统的这个核心组件负责执行程序。内核负责随时决定许多正在运行的程序中哪一个应该分配给处理器或处理器

Infiniband

InfiniBand (**IB**) is a ==computer networking communications standard ==used in high-performance computing that features very high throughput and very low latency. It is used for data interconnect both among and within computers. InfiniBand is also used as either a direct or switched interconnect between servers and storage systems, as well as an interconnect between storage systems. It is designed to be scalable and uses a switched fabric network topology. By 2014, it was the most commonly used interconnect in the TOP500list of supercomputers, until about 2016.1(https://en.wikipedia.org/wiki/InfiniBand#cite_note-down-1)

head-of-the line blocking

Head-of-line blocking (**HOL blocking**) in computer networking is a performance-limiting phenomenon that occurs when a line of packets is held up in a queue by a first packet. Examples include input buffered network switches, out-of-order deliveryand multiple requests in HTTP pipelining.

A switch may be composed of buffered input ports, a switch fabric and buffered output ports. If first-in first-out (FIFO) input buffers are used, only the oldest packet is available for forwarding. If the oldest packet cannot be transmitted due to its target output being busy, then more recent arrivals cannot be forwarded. The output may be busy if there is output contention.
（In computer science, **resource contention** is a conflict over access to a shared resource such as random access memory, disk storage, cache memory, internal buses or external network devices. A resource experiencing ongoing contention can be described as **oversubscribed**.）

Without HOL blocking, the new arrivals could potentially be forwarded around the stuck oldest packet to their respective destinations. HOL blocking can produce performance-degrading effects in input-buffered systems.

This phenomenon limits the throughput of switches. For FIFO input buffers, a simple model of fixed-sized cells to uniformly distributed destinations, causes the throughput to be limited to 58.6% of the total as the number of links becomes large.1(https://en.wikipedia.org/wiki/Head-of-line_blocking#cite_note-Karol-1)

One way to overcome this limitation is by using virtual output queues.2(https://en.wikipedia.org/wiki/Head-of-line_blocking#cite_note-2)

Only switches with input buffering can suffer HOL blocking. With sufficient internal bandwidth, input buffering is unnecessary; all buffering is handled at outputs and HOL blocking is avoided. This no-input-buffering architecture is common in small to medium-sized ethernet switches.

[补充材料]

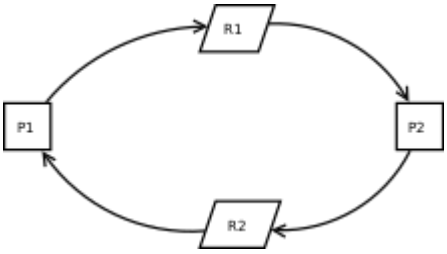
VOQ

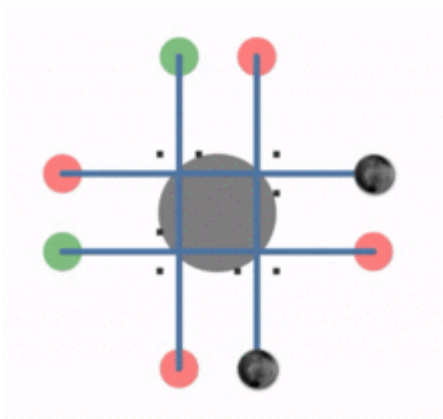
虚拟输出队列（VOQ）是某些网络交换机架构中使用的一种技术，其中不是将所有流量保留在单个队列中，而是为每个可能的输出位置维护单独的队列。它解决了被称为一线拦截的常见问题。1(https://en.wikipedia.org/wiki/Virtual_output_queueing#cite_note-1)

potential for deadlock

在并发计算中，**死锁**是指某些实体组中的任何成员都无法继续进行的任何情况，因为每个成员都在等待包括自身在内的另一个成员采取行动，例如发送消息或更常见的是释放锁。1(https://en.wikipedia.org/wiki/Deadlock#cite_note-coulouris-1)死锁是[多处理](https://en.wikipedia.org/wiki/Multiprocessing "多处理")系统、并行计算和分布式系统中的一个常见问题，因为在这些情况下，系统通常使用软件或硬件锁来仲裁共享资源并实现流程同步。2(https://en.wikipedia.org/wiki/Deadlock#cite_note-para_enclo-2)

在操作系统中，当进程或线程进入等待状态时会发生死锁，因为请求的系统资源被另一个等待进程持有，而另一个等待进程又在等待另一个等待进程持有的另一个资源。3(https://en.wikipedia.org/wiki/Deadlock#cite_note-Falsafi_Midkiff_Dennis_Dennis_2011_pp._524%E2%80%93527-3)如果一个进程由于它请求的资源被另一个本身正在等待的进程使用而无限期地无法改变其状态，那么该系统被称为陷入僵局。





RDMA transport livelock

"Livelock" redirects here. For the video game, see Livelock (video game).

A *livelock* is similar to a deadlock, except that the states of the processes involved in the livelock constantly change with regard to one another, none progressing.

The term was coined by Edward A. Ashcroft in a 1975 paper18(https://en.wikipedia.org/wiki/Deadlock#cite_note-18) in connection with an examination of airline booking systems.19(https://en.wikipedia.org/wiki/Deadlock#cite_note-19) Livelock is a special case of resource starvation; the general definition only states that a specific process is not progressing.20(https://en.wikipedia.org/wiki/Deadlock#cite_note-20)

Livelock is a risk with some algorithms that detect and recover from *deadlock*. If more than one process takes action, the deadlock detection algorithm can be repeatedly triggered. This can be avoided by ensuring that only one process (chosen arbitrarily or by priority) takes action.21(https://en.wikipedia.org/wiki/Deadlock#cite_note-21)

the NIC PFC pause frame storm

the slowreceiver symptom

VLAN [32] tags

VLAN技术要点主要有两点：

1.支持VLAN的交换机的内部交换原理；
2.设备之间(*交换机之间，交换机与路由器之间，交换机与主机之间*)交互时，*VLAN TAG的添加和移除*。

802.1Q VLAN只定义了数据帧的封装格式，即，在以太网帧头中插入了4个字节的VLAN字段。其主要内容为VLAN TAG，紧随其后的数据类型和802.1p报文优先级的标识。

以太网帧格式
| DMAC(6bytes) | SMAC(6bytes) | Ether-Type(2bytes) | DATA |

带VLAN TAG的以太网帧格式
| DMAC(6bytes) | SMAC(6bytes) | Ether-Type(0x8100) | VLAN(4bytes) | DATA |

VLAN TAG的格式
| PRI(3bits) | CFI(1bit) | TAG(12bits) | Ether-Type(2bytes) | DATA |

PRI：帧优先级，就是通常所说的802.1p。

CFI：规范标识位，0为规范格式，用于802.3或EthII。

TAG：就是我们通常说的VLAN ID

Ether-Type：标识紧随其后的数据类型。

PC：大部分的PC（专用的，或用于测试的除外）是工作在应用层的，缺省情况下是不支持(其实也不需要)VLAN TAG的。也就是说，PC发出的都是UNTAGED数据帧。

Router：路由器是支持VLAN TAG的。也就是说，路由器可以发出TAGED数据帧，也可以发出UNTAGED数据帧。需要说明的是，路由器是处理数据包的三层信息的，对于二层信息（包括VLAN信息），路由器只是检查其有效性，之后将其剥离。这个过程就是我们常说的'终结'，也就是说，路由器会终结掉报文的VLAN信息的。

Switch：以太网交换机。VLAN技术就是主要针对于交换机提出的，所以，在讨论VLAN概念时都是立足于交换机来讨论。很显然，交换机既支持收发TAGED数据帧，也支持收发UNTAGED数据帧。从严格意义上讲，引入VLAN后，交换机的行为不再是'透明传输'，因为数据帧经过交换机后可能发生了变化。

典型配置

交换机连接不同典型设备时的常用配置。

1). 连接PC
在通常情况下，PC只支持收发UNTAG的数据帧，所以，连接PC的端口只需要加入一个VLAN，而且，在该VLAN中的属性为UNTAG。

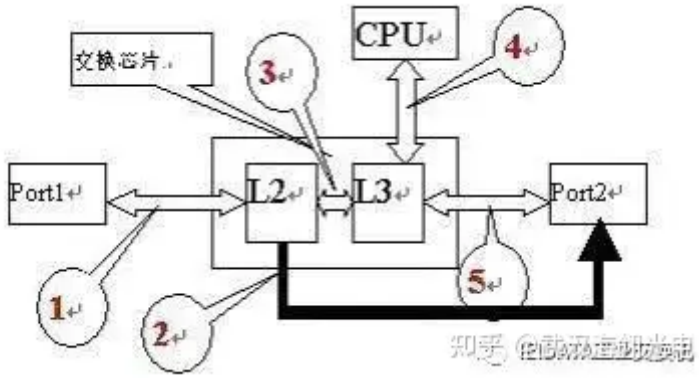
2). 连接Router
路由器既支持收发TAG数据帧，也支持收发UNTAG数据帧。通常情况下，不同的VLAN数据帧都能通过该端口与路由器互通。所以连接路由器的端口可以属于多个VLAN，而且，只能在一个VLAN中的属性是UNTAG的，在其他的VLAN中都是TAG的。

3). 连接Switch
也就是交换机的级联。通常情况下是不同性能的交换机进行级联。这种情况和连接路由器的情况基本相同。

交换机对VLAN的tag处理

所有能感知VLAN的交换机，报文在交换机内部转发过程中都是带Tag的。在交给交换芯片处理之前，或者交换芯片交给端口时会根据端口的设置添加或去掉Tag。
（内部恒带tag，对外是否有tag根据类型需求待定）

如果没有进行配置，默认所有的端口都处于Access模式。一般来说，对端口来说与VLAN相关的有三个属性，PVID、Tag Vlan、Untag Vlan。
1）PVID有且只有一个，Tag Vlan和Untag Vlan可能有一个，多个，也可以没有，但两个中至少有一个。
2）PVID的作用是：如果此端口收到一个Untag的报文，则交换机会根据PVID的值给此报文打上等于PVID的Tag，然后交给交换芯片处理；如果交换芯片要将一个报文从此端口发送，发现此报文的Tag值与PVID相同，则会将Tag去掉，然后从此端口发送出去。
3）Tag Vlan和Untag Vlan主要是用于报文发送的处理，如果交换芯片要将报文从此端口发送，发现报文的Tag在此端口的Tag Vlan中，且不等于PVID，则此报文将以Tag的形式发送出去，如果在Untag Vlan中，则以Untag形式发出去。如果报文的Tag既不在Tag Vlan中，也不在Untag Vlan中，则只有一种可能，交换芯片出毛病了。这就是交换机处理Vlan Tag的基本原则了。可以如图所示，即Tag处理只在交换芯片的进、出时刻。



这里做了简化，假设一个报文从Port1入，从Port2出，期间可能经过的环节如上所示的1－5。

报文的Tag处理主要在1、5

环节1：如果报文的是Untag的，则报文被加上Port1的PVID的tag，然后送给2层转发引擎L2。如果报文的Tag不是Port1所属的VLAN，则报文丢弃。

环节5：如果报文此时的Tag等于Port2的PVID，则报文去Tag，变为Untag的，如果报文的Tag在端口的Untag列表里，报文去掉Tag，变为Untag。如果报文的Tag不是Port2的PVID，但在端口的Tag列表里，报文Tag不变。

对于任意端口来说，只能是三种类型的一种：Access，Trunk，Multi。

【1】如果是Access，端口只能属于特定VLAN，只能有一个Untag的VLAN，且其等于PVID，没有Tag的VLAN。因此Access端口只能接收Untag的报文或所属的VLAN，即等于PVID的VLAN。出去的报文都是Untag的。

【2】如果是Trunk端口，端口只能有一个Untag的VLAN，且其等于PVID，有多个Tag的VLAN。端口可以接收在所有允许的VLAN和Untag的报文，如果是Untag的报文，则在PVID所在的VLAN转发。报文发送时，按上述规则转发，即可能是Tag的，也可能是Untag的。

【3】Multi端口是一种特殊的端口，任何VLAN可能属于此端口的Untag的，也可以属于Tag的。PVID可能等于Tag VLAN中的一个，也可能属于Untag VLAN中的一个。因此这里就有矛盾的是，如果PVID是Tag VLAN中的一个，转发时仍按PVID来处理，也就是上述的规则，先进行报文的Tag与PVID的比较，只要相等就给Untag。

可以再简要分析几种转发情况。

如果报文从Port1入，如果报文是Untag的，或者报文的Tag是Port1所属的VLAN，则报文是合法的。如果报文的MAC地址不等于端口的VLAN的MAC地址，则进行二层转发。如果L2引擎查找到目的MAC的表项，则按端口转发，否则在VLAN内广播。

如果报文的MAC地址等于端口的VLAN的MAC地址，且此VLAN的三层接口是UP的，则会进行三层转发。报文通过环节3到三层转发引擎，如果三层转发引擎匹配到了目的IP，则硬件直接转发，从环节5出，此时Tag已经变成出接口VLAN的Tag，然后在按照端口转发原则进行转发。因此，三层报文如果从Trunk端口出，可能是Tag的，也可能是Untag的，如果是Untag的，那肯定是报文出接口的PVID所对应的VLAN。

这里还有一个细节是交换机如何对协议报文的处理，最频繁的应该是ARP的请求报文，其余还有一些路由协议或其他协议报文。ARP报文的目的MAC是广播的，不等于VLAN的目的MAC，因此会将此报文在VLAN内广播，对所有VLAN来说，CPU也是其的一个端口。因此CPU是可以收到其他主机或网络设备对其IP的ARP请求，正确处理之后，其他主机或网络设备以后就可以使用此MAC进行三层通信了。

引入VLAN概念后，数据帧只在相应的VLAN进行交换。用通俗一点的话来讲，一个交换机被虚拟出了多个逻辑交换机，每一个VLAN内的端口都是一个逻辑上的交换机。用专业一点的话来讲，一个交换机被划分了多个不同的广播域，每一个VLAN内的端口，在同一个广播域内。

引入VLAN后的交换原理与传统的交换原理相比，并没有本质上的改变，同样遵循‘源MAC学习，目的MAC转发’的基本原则。唯一不同的是，学习和转发都只在同一个VLAN中进行，数据帧不能跨VLAN交换或转发。

如果收到的数据帧携带了VLAN信息 （通常称为'TAGED数据帧'，前面已经介绍了带VLAN TAG的以太帧格式），该VLAN信息中的VLAN TAG就是交换该帧的VLAN。

如果收到的数据帧没有携带VLAN信息（通常称为‘UNTAGED’数据帧），收到该帧的端口的PVID就是交换该帧的VLAN。

根据上面的原则，也定义了PVID的概念。当端口收到一个UNTAGED数据帧时，无法确定在哪个VLAN中进行交换，PVID定义了在这种情形下交换该帧的VLAN。从某种意义上讲，可以把PVID理解为端口的default VLAN。在支持VLAN的交换机中，每个端口都有一个PVID值，该值有一个缺省值，当然你也可以更改它。

交换机收发数据帧的处理总结

分几种情况讨论交换机的接收和发送处理：接收端口和发送端口在VLAN中属性；收到的数据帧是TAG的还是UNTAG的。

1). 端口接收到数据帧

a). 如果是TAG的数据帧，检查该接收端口是否在该VLAN(数据帧中所携带的VLAN TAG)中

- 接收端口在该VLAN中，则在该VLAN中根据交换原理(即，'源MAC学习，目的MAC转发'的原理)交换该数据包
- 接收端口不在该VLAN中，丢弃该数据帧

b). 如果是UNTAG的数据帧，检查该接收端口是否在某个VLAN中的属性是UNTAG

- 接收端口在某个VLAN中的属性是UNTAG的，则在该VLAN中根据交换原理交换该数据包
- 接收端口在任何VLAN中的属性都不是UNTAG的，丢弃该数据包

注：根据这个原理可知，一个端口最多在一个VLAN中的属性是UNTAG的，否则，收到一个UNTAG的数据帧之后，就无法确定在哪个VLAN中进行交换。其实，端口UNTAG所在的VLAN，

2). 端口发送数据帧

a).检查该端口在该VLAN(就是交换该数据帧的VLAN)中的属性

- 该端口在该VLAN种的属性是TAG的，发送的数据帧为TAG的数据帧
- 该端口在该VLAN种的属性是UNTAG的，发送的数据帧为UNTAG的数据帧

注：由于数据已经被交换到该端口，说明该端口肯定在该VLAN里。