RDMA核心概念与技术

RDMA核心概念与技术

The challenge for InfniBand’s designers was to create these channels between virtual address spaces which would be capable of carrying messages of varying sizes, and to ensure that the channels are isolated and protected. These channels would need to serve as pipes or connections between entirely disjoint virtual address spaces. In fact, the two virtual spaces might even be located in entirely disjoint physical address spaces – in other words, hosted by different servers… even over a distance.

这段话给出了RDMA区别于普通网络的核心技术点：首先，RDMA在设计之初，想要安全绕过（Bypasses）操作系统的参与，因此提出，应该在需要通信的两个应用程序（包括用户态应用程序和内核态应用程序）之间建立“通道”（channel），该通道具备携带任意大小数据的能力，并且该通道具备优秀的隔离性和保护性，在传统的网络中，隔离性和保护性由操作系统系统。RDMA建立的通道好比管道或者连接，这些管道或者连接能够作用于两个完全不相邻的虚拟地址空间。事实上，两个虚拟地址空间可能完全是两个不同的物理地址空间，也就是说，虚拟地址空间所对应的物理地址空间在地理上完全在不通过的服务器上。

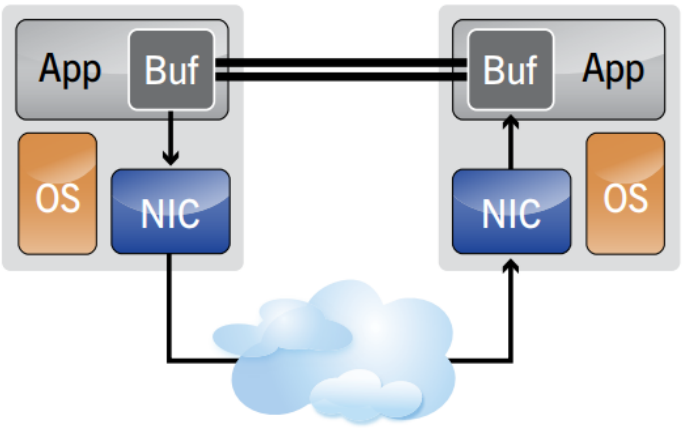

Figure 1: *InfiniBand creates a channel directly connecting an application in its virtual address space to an application in another virtual address space. The two applications can be in disjoint physical address spaces – hosted by different servers.*

从图中，我们能够看到，IB创建了一个channel直接连接了两个应用程序的虚拟地址空间，并且这两个应用程序完全可以是不相邻的物理地址空间。下图中，我们能够看到RDMA建立的channel，并且能够看到有两条线，它被称为Queue Pairs，分别由Send Queue和Receive Queue构成，并且应用程序A的Send Queue对应应用程序B的Receive Queue。应用程序将通过Queue Pairs访问IB提供的Messaging Services。

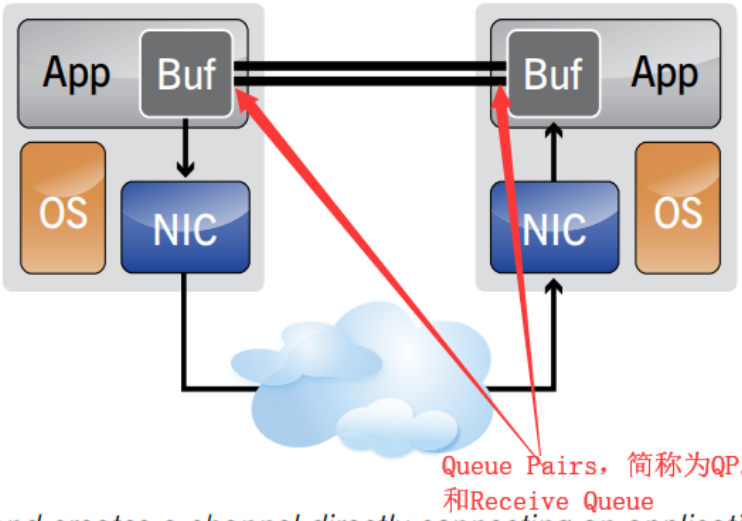

Figure 1: *InfiniBand creates a channel directly connecting an application in its virtual address space to an application in another virtual address space. The two applications can be in disjoint physical address spaces – hosted by different servers.*

In order to avoid involving the OS, the applications at each end of the channel must have direct access to these QPs. This is accomplished by mapping the QPs directly into each application's virtual address space.

为了避免调用操作系统，通道每一端的应用程序必须能够直接访问QPs，这通过将QPs映射进应用程序的虚拟地址空间来实现。

This is the essence of the InfniBand Architecture. It creates private, protected channels between two disjoint virtual address spaces, it provides a channel endpoint, called a QP, to the applications at each end of the channel, and it provides a means（n. 方法） for a local application to transfer messages directly between applications residing in those disjoint virtual address spaces. Channel I/O in a nutshell.

上面所述内容是IB体系架构的本质。IB会在两个不相邻的虚拟地址空间上创建一个私有的、受保护的通道，它提供一个被称为QP的通道端点，它为本地应用程序提供了一种在驻留在那些不相交的虚拟地址空间中的应用程序之间直接传输消息的方法。简称为通道IO。

Having established a channel, and having created a virtual endpoint to the channel, there is one further architectural nuance needed to complete the channel I/O picture and that is the actual method for transferring a message. InfniBand provides two transfer semantics; a channel semantic sometimes called SEND/RECEIVE and a pair of memory semantics called RDMA READ and RDMA WRITE. When using the channel semantic, the message is received in a data structure provided by the application on the receiving side. This data structure was pre-posted on its receive queue. Thus, the sending side does not have visibility into the buffers or data structures on the receiving side; instead, it simply SENDS the message and the receiving application RECEIVES the message.

前述内容已经建立了通道，并且为该通道创建了一个虚拟的端点（该端点被称为QPs）。接下来进一步的细节问题是：传递这些消息的真实可操作的方法是什么？（也就是怎么写代码去操作该通道及其对应的QPs？这是我的理解，如果理解有误，欢迎批评指正）。IB提供了两个传递语义，一种是channel语义（也被称为SEND/RECEIVE），另外一种是一对内存语义（也被称为RDMA READ和RDMA WRITE）。当在应用程序中使用chennel语义时，消息通过由接收端应用程序提供的数据结构接收，这个数据结构被提前预发布（pre-posted）在该应用程序的Receive Queue中。因此，发送方无法看到接收方的缓冲区或数据结构，发送方仅仅需要SEND这个message，而接收方仅仅需要RECEIVE这个消息即可（互相并不能事先知道对应发送或者接收的情况）。

The memory semantic is somewhat different; in this case the receiving side application registers a buffer in its virtual memory space. It passes control of that buffer to the sending side which then uses RDMA READ or RDMA WRITE operations to either read or write the data in that buffer.

内存语义不同于channel语义，接收方需要在它的虚拟地址空间注册一个buffer，接着接收方会将该buffer的控制权给发送方，然后通过使用RDMA READ或者RDMA WRITE操作去读或者写buffer中的数据。

A typical storage operation may illustrate the difference. The “initiator” wishes to store a block of data. To do so, it places the block of data in a buffer in its virtual address space and uses a SEND operation to send a storage request to the “target” (e.g. the storage device). The target, in turn, uses RDMA READ operations to fetch the block of data from the initiator’s virtual buffer. Once it has completed the operation, the target uses a SEND operation to return ending status to the initiator. Notice that the initiator, having requested service from the target, was free to go about its other business while the target asynchronously completed the storage operation, notifying the initiator on completion. The data transfer phase and the storage operation, of course, comprise the bulk of the operation.

一个典型的存储操作能够说明上述两种语义的差异。“initiator”想要存储一块数据，接着，它放置这块数据在虚拟地址空间的buffer中，并且使用SEND操作去发送一个send存储请求给“target”方。紧接着，“target”方使用RDMA READ操作从“initiator”方的虚拟buffer中去“拿 (fetch)”这块数据。一旦“target”方完成了拿数据的操作，它就会使用SEND操作去返回一个结束状态给“initiator”方。请注意，当“initiator”发出请求给target之后，“initiator”就恢复自由之身了，它能够做任何想做的事情，当target方异步完成数据的存储操作之后，它就会通知“initiator”方存储操作已经完成，当然，整个存储操作由大量的操作组成。

Included as part of the InfniBand Architecture and located right above the transport layer is the software transport interface. The software transport interface contains the QPs; keep in mind that the queue pairs are the structure by which the RDMA message transport service is accessed. The software transport interface also defines all the methods and mechanisms that an application needs to take full advantage of the RDMA message transport service. For example, the software transport interface describes the methods that applications use to establish a channel between them. An implementation of the software transport interface includes the APIs and libraries needed by an application to create and control the channel and to use the QPs in order to transfer messages.

作为IB结构的一部分，位于传输层（Transport Layer）正上方的是software transport interface。 该层包含QPs，请一定要牢记，QPs是RDMA实现消息传递服务必须被访问的结构。软件传输层也定义了一个应用程序需要充分利用RDMA消息传递服务的所

有的方法和机制。例如，软件传递接口描述了不同应用程序在其虚拟地址空间建立channel的全部方法。软件传输接口包含了创建和控制channel以及使用QPs的所有APIs和Libs。下图是IB Messaging Service架构图。我们可以看到最上层是software（S/W）transport interface。OSI 模型包含7层，IB的Messaging Service模型包含5层，自上而下分别是：软件传输接口、传输层、网络层、数据链路层、物理层。其中software transport interface是IB结构区别于其它网络架构最显著的部分。

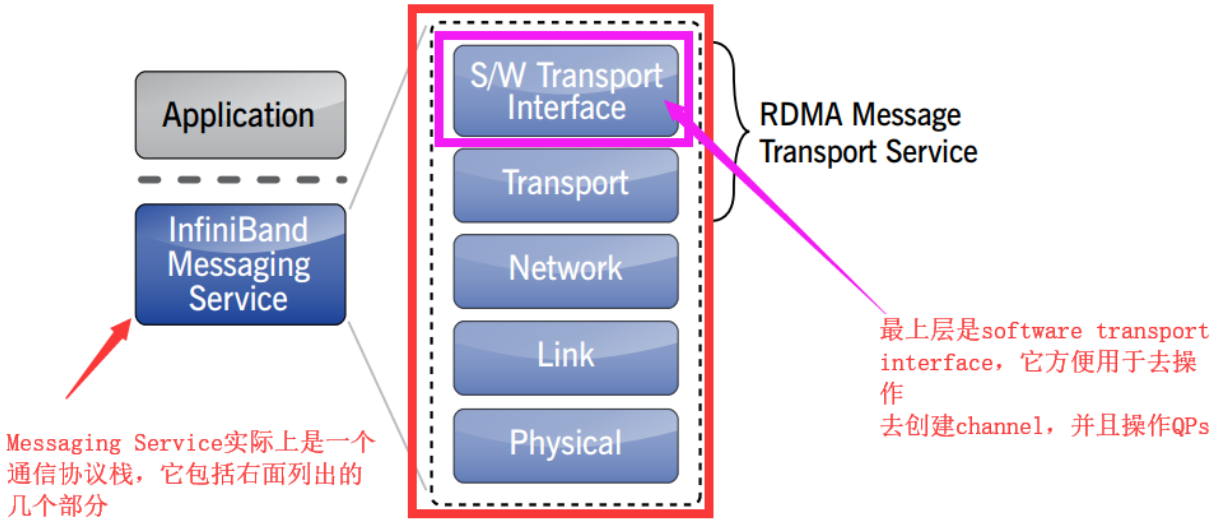

Figure 2: *The InfiniBand Architecture provides an easy-to-use messaging service to applications. The messaging service includes a communications stack similar to the familiar OSI reference model.*

OSI Model			
Layer		Protocol data unit (PDU)	Function[3]
Host layers	7. Application	Data	High-level APIs, including resource sharing, remote file access
	6. Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5. Session		Managing communication sessions, i.e. continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4. Transport	Segment, Datagram	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media layers	3. Network	Packet	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2. Data link	Frame	Reliable transmission of data frames between two nodes connected by a physical layer
	1. Physical	Symbol	Transmission and reception of raw bit streams over a physical medium

the combination of InfniBand's transport layer together with the software transport interface is better thought of as a Remote Direct Memory Access (RDMA) message transport service. The entire stack taken together, including the software transport interface, comprise the InfniBand messaging service.

IB的transport layer和software transport interface，共同组成了IB的Messaging Service。结合在一起被看做一个RDMA 消息传递服务。整个stack聚集在一起，也包括software transport interface，共同组成了IB的Messaging Service。

How does the application actually transfer a message? The InfniBand Architecture provides simple mechanisms defined in the software transport interface for placing a request to perform a message transfer on a queue. This queue is the QP, representing the channel endpoint. The request is called a Work Request (WR) and represents a single quantum of work that the application wants to perform. A typical WR, for example, describes a message that the application wishes to have transported to another application.

应用程序之间怎么实现消息传递呢？IB的体系架构提供了一个简单的机制：它被定义在software transport interface中，通过放置一个消息传递请求在queue中，这个queue被称为QP，它表示channel的端点。发出的请求被称为Word Request (WR)，它表示应用程序想要执行的操作。

This messaging service is a distinctly different service than
is provided by other traditional networks such as TCP/IP, which are adept at moving strings of bytes from the operating system in one node to the operating system in another node。

消息服务是IB架构区别于其它传统网络架构核心内容，普通的网络擅长通过操作系统从一个端点传递到另外一点端点，并且这个传递过程传递字节类型的字符串。而IB的结构是传递任意长度类型的消息。因此，我们说IB面向消息 (message-oriented)的协议。

InfniBand on the other hand, delivers a complete message to an application. Once an application has requested transport of a message, the InfniBand hardware automatically segments the outbound message into a number of packets; the packet size is chosen to optimize the available network bandwidth. Packets are transmitted through the network by the InfniBand hardware and at the receiving end they are delivered directly into the receiving application’s virtual
buffer where they are re-assembled into a complete message. Once the entire message has been received the receiving application is notifed. Neither the sending nor the receiving application is involved until the complete message is delivered to the receiving application’s virtual buffer.

另一方面，IB传递一个完整的消息给某个应用程序。一旦一个应用程序发出传递信息的requested后，IB的硬件将会自动将传送出来的数据分段成一系列的包，**包的大小可以被选择，这样能够优化网络带宽。**包被IB的硬件网络传递，在接收端，包被重组成一个完整的信息并被直接传递进入应用程序的虚拟buffer中。一旦整个消息被接受，接

受端将发出“完成”通知。整个消息传递的过程，发送方和接收方都没有参与，直到整个消息被完整的传输到接收方的虚拟buffer中发送方和接受方才会参与发送“完成”通知和接收“完成”通知。

说明

本文的内容大部分翻译自：Introduction to InfniBand™ for End Users 文档。

参考资料

1. https://zcopy.wordpress.com/
2. https://opensourceforu.com/2016/09/fundamentals-of-rdma-programming/