SoftDC: software-based dynamically connected transport

Jiwoong Park, Yongseok Son, Heon Young Yeom & Yoonhee Kim

Cluster Computing
The Journal of Networks, Software Tools and Applications

ISSN 1386-7857
Volume 23
Number 1

Cluster Comput (2020) 23:347-357
DOI 10.1007/s10586-019-02926-0

Springer

Cluster Computing (2020) 23:347–357
https://doi.org/10.1007/s10586-019-02926-0

SoftDC: software-based dynamically connected transport

Jiwoong Park[1] · Yongseok Son[2] · Heon Young Yeom[1] · Yoonhee Kim[3]

Received: 2 January 2019 / Accepted: 9 March 2019 / Published online: 19 March 2019

Abstract
RDMA is increasingly becoming popular not only in HPC but also in data centers where high throughput and low latency are critical requirements. RDMA supports several types of transports, each of which has different characteristics, so that users can choose the right one to meet their requirements. Reliable connected (RC) transport has advantages on usability but disadvantages on scalability while unreliable datagram (UD) transport is scalable but hard to use. Dynamically connected (DC) transport has been newly introduced to address these limitations when using one of the existing transports while delivering both usability and scalability. However, despite all of these merits, DC transport is not yet generally adopted in the related fields due to hardware dependency. To eliminate the hardware dependency, in this paper, we design and implement SoftDC, a totally software-based DC transport. SoftDC uses the basic RDMA primitives to emulate the behavior of DC transport; UD transport for connection and RC transport for data transfer. We build and evaluate a SoftDC transport-based application to prove its effectiveness compared with RC transport-based one. Our experimental results show that our scheme has potential to provide the advantages of both RC and UD transports although our prototype has large connection establishment costs that can be amortized across a large data transfer.

Keywords RDMA · Queue pair · DCT

1 Introduction

Interconnect speed among the nodes in a cluster has been an important factor in high-performance computing (HPC), especially for cluster architecture. In such environment, remote direct memory access (RDMA) is the key to high performance with low CPU overhead, moving the responsibility of transferring data between nodes from the CPU to the network adapter. Since new standards, such as Internet wide-area RDMA protocol (iWarp) and RDMA over Converged Ethernet (RoCE), were developed to work with the commodity Ethernet switches and routers, RDMA has become widely used from HPC to datacenters.

Current RDMA implementation provides several communication transports with different characteristics. A TCP-like reliable, connection-oriented (RC) transport and a UDP-like connectionless unreliable datagram (UD) transport are most commonly used. The strengths of the former lie in high performance with full communication support while the latter has advantages on scalability.

Since both transports have each advantages and disadvantages, there has been several efforts to apply a hybrid approach of them in application level [2, 3]. MVAPICH-Aptus [2] is a multi-transport MPI design that dynamically

A preliminary version [1] of this article was presented at the 6th International Workshop on Autonomic Management of high performance Grid and Cloud Computing (AMGCC'18), Trento, Italy, Sep. 2018.

✉ Yoonhee Kim
yulan@sookmyung.ac.kr

Jiwoong Park
pjbear@snu.ac.kr

Yongseok Son
sysganda@cau.ac.kr

Heon Young Yeom
yeom@snu.ac.kr

[1] Department of Computer Science and Engineering, Seoul National University, Seoul, South Korea

[2] School of Computer Science and Engineering, Chung-Ang University, Seoul, South Korea

[3] Department of Computer Science, Sookmyung Women's University, Seoul, South Korea

chooses one out of the transport pool based on the defined rules, considering the message size and the number of ranks in the MPI job. Jithin et al. [3] have designed and implemented a hybrid RC/UD transport in the Unified Communication Runtime (UCR) [4], which is designed for HPC and big data middleware. It preferentially uses RC transports until the specified threshold is reached, then uses UD transports to limit the memory footprint. However, all their approaches require significant changes in the design of applications.

To provide a scalable high-performance RDMA transport, Mellanox has recently introduced dynamically connected (DC) transport, which takes the best of both RC and UD transports. Since DC transport is a transport itself unlike the aforesaid approaches, it is easier for the existing RDMA applications to enjoy the high functionality and performance of the RC transport and the scalability of the UD transport only with small changes in the application. This is like the case where the TCP-based application can be much simpler to implement than UDP-based one. Despite its superiority, however, DC transport has not yet entirely replaced the existing RDMA transports due to one major reason; vendor-specific hardware is required to use DC transport. Currently, only Mellanox Infiniband adapter card ConnectX-IB and Ethernet adapter cards above ConnectX-4 support DC transport.

In order to make DC transport generally available to all RDMA-based solutions, in this paper, we design and implement SoftDC, a totally software-based DC transport, eliminating hardware dependency. SoftDC utilizes both RC and UD transports to emulate the behavior of DC transport: UD for connection, RC for RDMA/atomic operations and data transfers. We have built a SoftDC prototype as an user library based on the user-level RDMA libraries such as *librdmacm* and *ibverbs*.

To prove the effectiveness of SoftDC, we conduct several experiments to test the overhead of it, comparing with RC QP. The results show that although connection establishment cost of SoftDC can be problematic for small data transfer, there are some trade-offs to alleviate this overhead, allowing SoftDC to provide the advantages of both RC and UD transports.

Our contributions are fourfold as follows:

- We show that DC transport can be implemented in software without vendor-specific hardware support while providing the same functionality as hardware DC transport.
- With extensive experiments, we show that SoftDC has potential to have good balance between performance and scalability.
- We demonstrate that SoftDC transport can be easily ported to existing RDMA applications with little effort.
- We propose a new use case of DC transport to extend the separation of control and data plane to the existing RC QP-based applications in the proxy architecture.

The rest of the paper is organized as follows: In Sect. 2, we provide the background information of RDMA. In Sect. 3, we explain the design and implementation of our scheme. In Sect. 4, we introduce the use case of SoftDC. We evaluate our prototype in Sect. 5. Section 6 discusses related works. We conclude and discuss future work in Sect. 7.

2 Background

RDMA enables direct memory access from the memory of a host into that of a remote host without the involvement of CPU. In this section, we introduce the background of the RDMA communication model and RDMA transports. We also perform a simple RDMA benchmark test to see the relationship between the number of queue pairs (QPs) and the performance.

2.1 RDMA communication model

In RDMA, a queue pair (QP) is conceptually similar to TCP/UDP socket which represents endpoints for sending and receiving messages. QP consists of Send Queue (SQ) and Receive Queue (RQ). When a user post send/receive work requests (WR) into QP, these WRs are placed into SQ/RQ depending on the request type and then they are processed by RDMA-enabled NIC (RNIC). In order to receive the message, a buffer has to be posted to the receive-side QP before sending messages to that QP. A shared Receive Queue (SRQ) allows for buffers to be posted to a single receive queue and to be shared by multiple QPs, thus preventing wasting memory on the receive buffers that would have been prepared for incoming messages on every QP.

There are two types of RDMA operations (also called a verb): channel semantic (two-sided verb) and memory semantic (one-sided verb). Two-sided verbs (SEND and RECV) require the involvement of the CPU at both sides (receiver and sender), providing traditional send/receive semantics. On the other hand, for one-sided verbs (READ, WRITE, and ATOMIC) the receive-side CPU is not aware of the operation once memory regions are registered by the receiver to be remotely accessible.

2.2 Default transport types

In RDMA, there are three transports depending on reliability and connection orientedness: Reliable Connection

(RC), Unreliable Datagram (UD), and Unreliable Connection (UC). We will discuss only the first two transports because UC transport is rarely used. Table 1 summarizes the characteristics of each transport type.

2.2.1 Reliable connection transport

RC transport is the most widely used one due to its rich functionality and ease of usability. Using RC transport, one can send any message whose size is smaller than 2 GB (depends on RNIC) without worrying about message segmentation and reliability. Large messages are automatically fragmented into MTU-sized packets and reassembled to original one by RNIC. Additionally, RC is the only transport that fully supports one-sided verbs including READ, WRITE, and ATOMIC, which enable ultra-low latency by bypassing the remote CPU.

As it is connection-oriented, a connection between two QPs must be exclusively established to communicate with each other. This one-to-one requirement of RC transport greatly limits the scalability for large cluster. When a thread wants to communicate with N remote nodes, it has to create N QPs for each remote node. Figure 1a shows an example where three clients communicate with one server using RC QP. For this case, the server needs three RC QPs in total. Due to the limited memory of RNIC to cache QP states, using too many QPs can cause cache thrashing, leading to performance degradation [5].

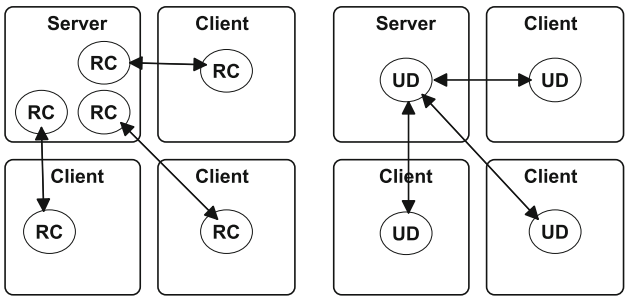
2.2.2 Unreliable datagram transport

Compared with RC transport, UD transport supports fewer features. Since maximum message size is limited by MTU and there are no reliable packet delivery guarantees in UD, message segmentation and reliability must be handled in application level, thus leading to more programming effort. Furthermore, none of the one-sided verbs are supported in UD transport, unlike RC transport.

However, despite all of these shortcomings, UD is superior to RC when it comes to scalability. Unlike RC transport where every QP has an address vector that describes the routing information to the remote node, the address vector in UD transport is defined in every send-request. To send messages to the remote QP, UD requires remote QP number (QPN) and remote QP key (QKEY) as well as address handle (AH) that is the implementation of an address vector. This makes it possible that one UD QP can communicate with multiple remote UD QPs, resulting in much better scalability. Figure 1b shows an example where three clients communicate with one server using UD QP. For this case, the server needs only one UD QP in total.

Table 1 The characteristics of each QP type

Characteristics	UD	UC	RC	DC
SEND	O	O	O	O
RDMA WRITE	X	O	O	O
RDMA READ	X	X	O	O
RDMA ATOMIC	X	X	O	O
Connection type	1:N	1:1	1:1	1:N
Maximum message size	MTU	2GB	2GB	2GB



(a) RC Scalability **(b)** UD Scalability

Fig. 1 Scalability

2.3 Vendor-specific transport

Mellanox has introduced DC transport to provide the reliability of RC transport as well as the scalability of UD transport. As of now, DC transport is only supported in Mellanox network adapters.

2.3.1 Dynamically connected transport

As the name indicates, DC transport allows a QP to dynamically (re)connect to the remote QP when needed and disconnect after data transmission has been completed. Since DC transport internally uses reliable type QPs, all one-sided verbs can be used when using DC. Additionally, the dynamic connectivity of DC transport makes the size of the cluster irrelevant to connection resource usages needed for communications, thus achieving similar scalability as UD transport.

There are two new different QP types in DC transport: DC initiator (DCI) and DC target (DCT). A single DCI can send messages to multiple targets like UD QP, and a DCT can receive messages from multiple DCIs at the same time. The usage model of DC is very similar to UD except that DC distinguishes between an initiator QP and a target QP. DC usage model is as follows. A user creates a DCT using DC access key which is defined by the user then creates DCIs depending on the needs of concurrency of data transmission. AH, DCT number, and DC access key are

required to identify the target DCT when sending messages using DCI.

In spite of its superiority, DC transport has not been generally adopted yet in RDMA-based software solutions. The need for vendor-specific hardware remains a limitation.

3 Software-based dynamically connected transport

SoftDC is a totally software-based DC transport for Ethernet, allowing the existing RDMA-based solutions to enjoy high performance and scalability of DC transport without any hardware support. Theoretically, every Ethernet network adapter can use RDMA without hardware support owing to the existence of Soft RoCE, which is a software implementation of RoCE included in the upstream kernel. (Note that DC transport is not currently supported in Soft RoCE as well.) To completely eliminate hardware dependency, we focus on how to design a software layer on top of the given RDMA primitives. In this section, we explain the system components and the overall process of SoftDC.

3.1 System components

SoftDC consists of four parts: (1) DCI and DCT as RDMA primitives, (2) *Automatic connection handler* to handle new connections, (3) *Connection table*, which is an in-memory key-value store to store connection information. (4) APIs to be used for building a SoftDC-based application.

3.1.1 DC Initiator and DC target

Like the existing DC transport model, SoftDC uses DCI to initiate data transfer and DCT to handle incoming data. SoftDC exploits the existing UD and RC transports to imitate the DC transport.

Each of DCI and DCT is composed of UD and RC QPs. To give software the illusion of using connectionless transport with full RDMA operation support, UD is the base QP to be exposed to users while RC QPs are completely hidden from users. In SoftDC, UD QP is only used to exchange the connection messages when a connection is needed, whereas RC QP is used to transfer the actual messages given by the user. In other words, any messages from the user are not delivered via UD QP but RC QP. Note that because UD is connectionless transport we can send connection messages to any remote UD QP if and only if we have AH, QPN, and QKEY of the target UD QP. Additionally, since the actual data transfer goes through RC QP all one-sided RDMA operations are freely used in SoftDC. DCI and DCT internally maintain and manage the buffers and other RDMA primitives to be used for the implicit connection handling.

3.1.2 Automatic connection handler

DCT in SoftDC creates an *automatic connection handler* thread at initialization that is a background thread to handle the incoming *connection invitations*. *Connection invitation* contains the information to reach the sender and only the receiver of the connection invitation can connect to the sender.

This thread polls the completion queue (CQ) of the UD QP in DCT, waiting for the connection invitation from the remote DCI. When it receives an invitation, it creates a new RC QP, then connects to the sender using the information contained in the invitation. After the connection has been established, the related information is stored in the *connection table* to be reused later.

3.1.3 Connection table

Connection table is a simple key value store to store the connection information as key-value pairs in memory. It is created with the initialization of a DCT and shared by multiple DCIs and the DCT. A key is a combination of *Global Unique Identifier(GID)* and the UD QPN of the target DCT to uniquely identify to which remote DCT we have connected while a value consists of the local RC QP connected to the target RC QP and its connection state to be used for data transfer.

Because a single DCT must have capable of handling concurrent incoming data transfers, we use the *connection table* to internally maintain the connection state at least until the data transfer is complete. By default, the *connection table* can hold 16 entries, which means 16 concurrent data transfers are possible, while this can be configure when initialized. Unlike the hardware DC transport, SoftDC does not immediately disconnect the connection after data transfer is complete, it rather keeps connections open for reuse. The stored information could accelerate the data transfer when a user wants to send messages to the same target by eliminating the connection time.

3.1.4 APIs

SoftDC provides a set of common APIs for applications to utilize the software-based DC transport. Due to the similarity between the APIs of UD and DC, we can easily create the APIs of SoftDC by wrapping the APIs of UD transport. Note that UD QP in SoftDC is exposed to user so

that user can use it as if it is a connectionless transport with full RDMA support.

Table 2 lists DC's APIs and SoftDC's APIs. SoftDC's APIs have their counterparts in the native DC APIs. They are almost the same except that creating a DCI requires a DCT to share the *connection table* included in the DCT data structure. *ibv_sdc_create_dct*() and *ibv_sdc_create_qp*() are used to create DCT and DCI respectively, allocating the internal resources while *ibv_sdc_destroy_dct*() and *ibv_sdc_destroy_qp*() are used to destroy DCT and DCI respectively, cleaning up and freeing the allocated resources. Using *ibv_sdc_post_send*(), users can send any RDMA operations to the target including all one-sided verbs with UD-like interface requiring only remote AH, QPN, and QKEY.

We have added two new APIs into the our previous work [1]. Having examined the end-to-end latency of SoftDC when sending the first message from client to server, which will be detailed in Sect. 5.2, we find that most of the latency comes from creating an AH in server-side, which is needed for reply. In general, however, servers already have AHs for clients in application-level, so that servers reply back to the sender. Therefore, this duplicate creation of AH can be avoided if there is an API allowing user to create AH and automatically store it into the *connection table*. *ibv_sdc_create_ah*() does the job while *ibv_sdc_destroy_ah*() is responsible for destroying of the created AH.

3.2 Overall process

No matter which RDMA operations users request, SoftDC undertakes the responsibility of delivering them to the target while providing scalability without complex application design. Figure 2 shows the overall process of SoftDC. The detailed explanation how it works is as follows.

When a user posts a send request using the SoftDC send API that additionally requires DCI, AH, QPN, and QKEY as arguments, SoftDC first checks if the connection between the local node and the target node has already been made, looking up the *connection table*. If the matching key exists and the connection is still alive, SoftDC easily posts the user's send request to the RC QP in the corresponding value. If there is no connection found in *connection table*, SoftDC simply creates a new RC QP. After preparing the RC QP, SoftDC makes it listen and then sends a connection invitation to the target DCT using the given AH, QPN, and QKEY. Having waited for the RC QP being connected, SoftDC stores the connection information such as RC QP into the *connection table* for later use and then post the original message that the user wants to send to the RC QP.

Since SoftDC uses the UD transport for connection messages, which is an unreliable transport, some connection messages may be dropped in a very congested environment. Currently, SoftDC prototype does not guarantee the reliable delivery of the connection messages but it can be implemented in a SoftDC layer by resending the connection messages if no replies are received from the target after the specified timeout. We leave it for future work.

4 SoftDC use-case: proxy architecture

One-sided RDMA verbs supports the separation of the control plane from the data plane by design. It involves kernel only on the control path, but accesses the RNIC directly from user space on the data path. For example, the memory region has to be pre-registered by user to be directly accessed by RNIC while data transfer is handled by RNIC once user initiates the request. This characteristics can be used to extend the separation of control and data planes to the existing system that does not support such separation.

However, one problem still remains; one-sided RDMA verbs is only supported in connection-oriented transport where user must explicitly establish and terminate the connection. For example, using connection-oriented transport, in a proxy architecture where a client communicates with a server through a proxy, all data transfer between client and server goes through the proxy if the protocol dose not support the separation of control and data planes.

Table 2 DC APIs and SoftDC APIs

DC API	SoftDC API	Explanation
ibv_exp_create_dct	ibv_sdc_create_dct	Create a DC target object
ibv_exp_destroy_dct	ibv_sdc_destroy_dct	Destory a DC target object
ibv_exp_create_qp	ibv_sdc_create_qp	Create a DC initiator object
ibv_destroy_qp	ibv_sdc_destroy_qp	Destroy a DC initiator object
ibv_create_ah	ibv_sdc_create_ah	Create an Address Handle object
ibv_destroy_ah	ibv_sdc_destroy_ah	Destroy an Address Handle object
ibv_exp_post_send	ibv_sdc_post_send	Post work requests to QP

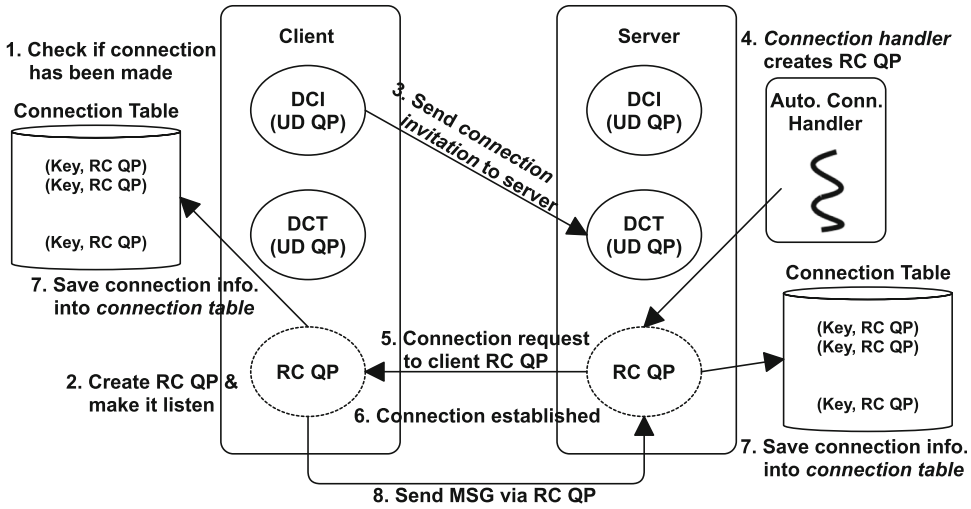


Fig. 2 The overall process of SoftDC

This results in unnecessary data movement and waste of network bandwidth. Since the client is unaware of the server's presence in the proxy architecture, the client cannot explicitly connect to the server.

Fortunately, DC transport supports one-sided RDMA verb, being a connectionless transport (on the user's point of view). Having similar semantic to DC transport, SoftDC can be used to provide the capability of separating control and data planes to the existing RC QP-based RDMA applications while minimizing modifications to the applications. In the SoftDC-based design, only the proxy needs to be modified to bidirectionally forwards the information of the memory region to other sides on the control path. On the data path, leveraging the forwarded information without knowing whose information it is, the client can directly access the remote memory region of the server. Note that SoftDC automatically internally handles the connection.

5 Evaluation

In this section, we evaluate the effectiveness of SoftDC. In this work, we focus on the overhead imposed by SoftDC because we do not have enough RNIC-equipped servers to produce the scalability issue discussed in Sect. 2. However, we believe that SoftDC would be scalable because it provides the dynamic connectivity like the native DC transport, making the size of the cluster irrelevant to connection resource usage.

Since our RNIC does not support the hardware DC transport, we are unable to directly compare the performance of SoftDC with the native DC. We instead compare the performance of the SoftDC-based applications with it of the RC-based applications to investigate its overhead.

5.1 System configuration

5.1.1 System setup

Unless specified otherwise, our experimental evaluations are conducted on two servers each with two 12-core Intel Xeon E5-2650 2.2 GHz, 160 GB of memory, Samsung 850 Pro SSD, and Mellanox ConnectX-3 40GbE RNIC, which does not support DC transport. The machines are connected via Mellanox MSX1012B-2BFS 40GbE Ethernet switch.

Since using RDMA in kernel-level is quite different from using it in user-level due to their different APIs, we write a user-level library and a kernel module for SoftDC. The user-level library is based on *librdmacm* and *ibverbs* libraries, whose version is 13-7.el7, which came with CentOS 7.4 while the kernel module is built against Linux kernel 4.12.4.

5.2 Micro benchmark

To measure the overhead imposed by SoftDC, we write a simple user-level RDMA client-server application where a client sends a message to a server and the server sends a reply back to the client, repeating this process one more time. Assuming that the client and the server already have each other's information to reach such as AH, QPN, and QKEY, we measure the client-perceived latency that each message exchange takes.

The application has three versions: RC, SoftDC, and SoftDC-Opt. RC is the RC-based application while SoftDC is the SoftDC-based one. SoftDC-Opt is the optimized version of the SoftDC-based application that leverages the new APIs for creating and destroying AHs, thus avoiding the necessity of the duplicate creation of AH, as discussed in Sect. 3.

5.2.1 Results

Figure 3a shows the results of the first message exchange obtained by varying the message size. Note that we use log scale on y-axis to show the wide range of latency for various message sizes. We can see that the performance gap between RC and SoftDC is rather huge for the first message while the latency for the second message exchange is almost the same for both RC and SoftDC transports, as shown in Fig. 3b. This significant difference comes from the fact that the RC-based one has established a connection between the client and the server before the time measurement while the SoftDC-based one defers the connection until the data transfer really occurs. We find that the results of two transports would be the same if we included the first connection time in the latency measurement. Once the first data transfer is complete, the SoftDC-based one can simply look up the connection table for the connected RC QP without generating overheads for connection.

To further investigate the main cause of the connection overhead in SoftDC, we examine the end-to-end latency of SoftDC when sending the first 1 KB message from client to server. Given the latency breakdown shown in Fig. 4, we can see that creating AH in server and establishing a connection between RC QPs accounts for almost all the first-time connection overhead of SoftDC. The cost of creating AH in server, which contributes 35% of the total latency, can be avoided if the application and SofDC library can share the created AHs. With help of the new APIs of SoftDC including *ibv_sdc_create_ah*() and *ibv_sdc_destroy_ah*(), we could eliminate this avoidable overhead. The result of SoftDC-Opt shows the result of the optimization. The first-time connection overhead of SoftDC-Opt is cut down to almost half of it of SoftDC. The gain from the optimization decreases as the message size increases because the relative proportion of the connection overhead in the end-to-end latency becomes smaller compared to the data transfer time. The connection overhead is still quite huge even with SoftDC-Opt. Current SoftDC prototype uses the RDMA Communication Manager (CM) to establish communication between two servers. We may further optimize the connection establishment process by directly exchanging the information needed for QPs to be connected. We leave it for future work.

5.3 FIO benchmark on NFS

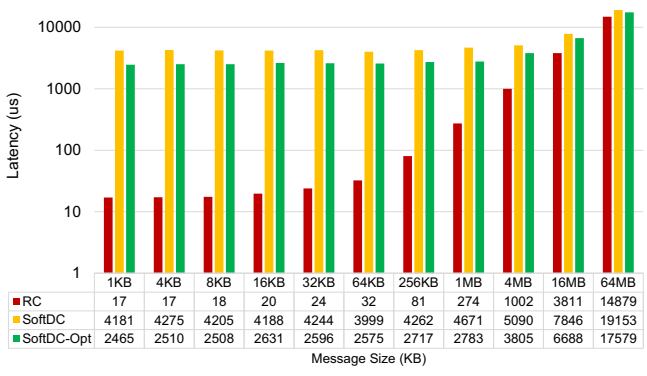
In order to measure the overhead of the SoftDC kernel module and to demonstrate that SoftDC can be easily ported to the existing RDMA applications, we have applied SoftDC into NFS server and client. Network file system (NFS) is a distributed file system that allows a server to share directories and files with multiple clients over a network [6]. NFS is implemented in the kernel layer, supporting only RC transport among RDMA transports.

Our SoftDC integration for NFS requires only about 50 lines of changes excluding SoftDC itself, showing its great portability. The changes include the addition of the DCI and DCT to the existing NFS data structures and the replacement of the existing RDMA APIs by SoftDC APIs.

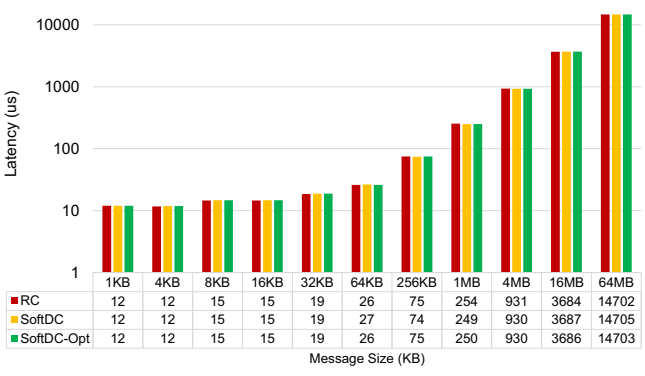
We use *fio-3.12* [7], which is an open-source synthetic IO benchmark tool, to measure the performance of the two versions of NFS. Varying the I/O size, we run four workloads of FIO on the NFS volume: sequential read, sequential write, random read and random write. Each of workloads is ran with 8 threads, 16 I/O depth, direct I/O mode and the libaio engine. Each thread generates 1GB I/O.

5.3.1 Results

The results are shown in Fig. 5. Comparing RC and SoftDC transport, we do not find much difference in the I/O performance for all workloads. This is because the connection



(a) 1st Message



(b) 2nd Message

Fig. 3 Latency of micro-benchmark

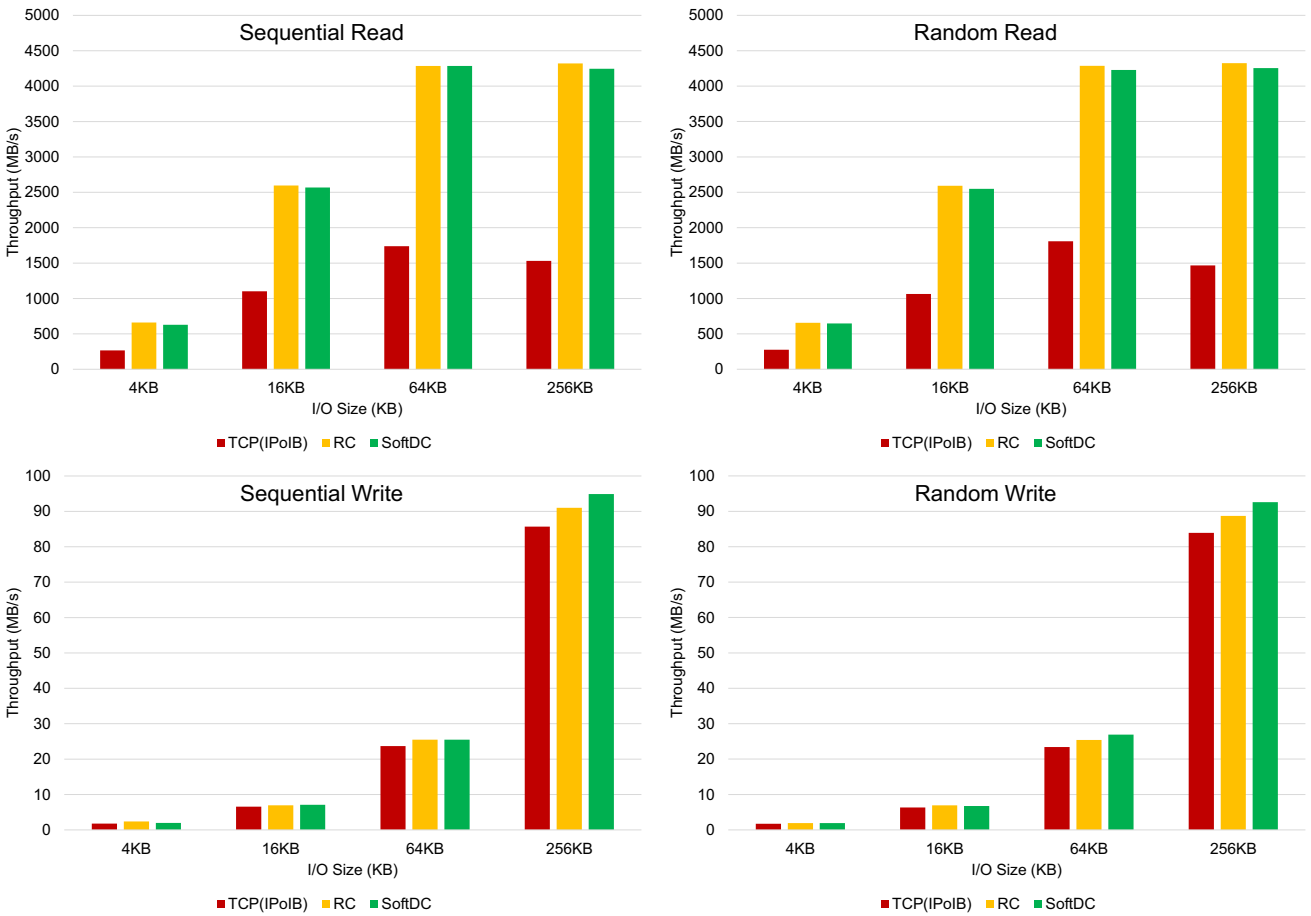


Fig. 4 Latency breakdown of SoftDC-based micro-benchmark for message size 1 KB

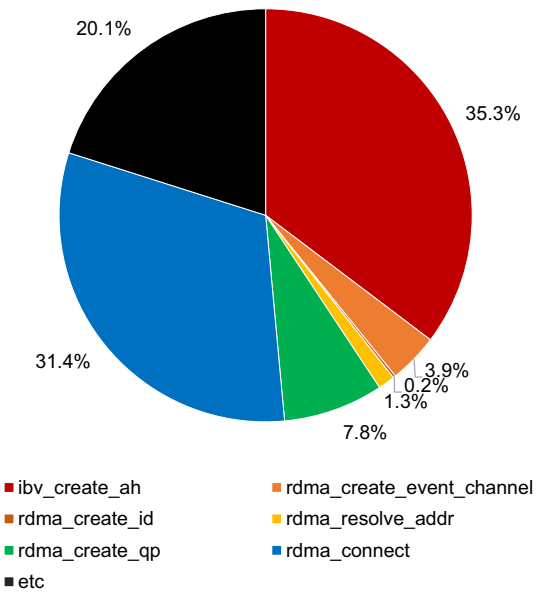


Fig. 5 FIO benchmark results on NFS with different transports: TCP(IPoIB), RC, and SoftDC

between NFS client and NFS server is established when a client mounts an NFS volume. Therefore, SoftDC acts like RC when doing I/O, adding only negligible overhead to look up the connection table.

We also add the result when using TCP(IPoIB) to compare. For the read workloads, RDMA transports deliver performance as high as twice that of the TCP(IPoIB) while the write performance is almost the same. Since in RDMA transports write requests are internally handled as RDMA READ which requires more round trip time than sending data directly to the NFS server. Also note that although we use direct I/O mode to bypass NFS client caching layer, NFS server may still cache the file contents in the kernel page cache. Since we mount the NFS volume with synchronous mode the NFS server replies to NFS clients only after the data has been written to the storage. These explain the higher performance of the read workloads than the write workloads.

5.4 SoftDC use-case: proxy architecture

As discussed in Sect. 4, SoftDC can be used to extend the separation of control and data planes in the proxy

architecture, where a client communicates with a server through a proxy. To demonstrate the effectiveness of the SoftDC-based proxy architecture, we write a simple benchmark where a client sends the information of its memory region to a server and the server performs RDMA WRITE to the memory region and notifies the client of completion. The benchmark repeat this one more time like the previous micro-benchmark. A proxy is between the client and the server, forwarding the request and the reply to other sides. The benchmark has two versions that simulate the RC QP-based and the SoftDC-based proxy system, respectively. The overall process of each version is shown in Fig. 6. We run this benchmark, varying the amount of data transfer.

5.4.1 Results

Figure 7a shows the client-perceived latency of the first request. Note that we use log scale. For a data transfer smaller than 16 MB, similar to the micro-benchmark test, SoftDC-based system takes longer latency for the request to be handled due to the first-time connection overhead, which offsets the benefits gained by eliminating unnecessary data movement. However, since this connection overhead is rather constant, regardless of the data size, the situation is reversed for larger data transfer. The gains from the reduction in data movements become larger than the cost of the connection establishment.

Figure 7b shows the client-perceived latency of the second request. As we expected, SoftDC-based system shows better latency for every size of data than RC QP-based system because the connection between the client and the server has already been established at this point.

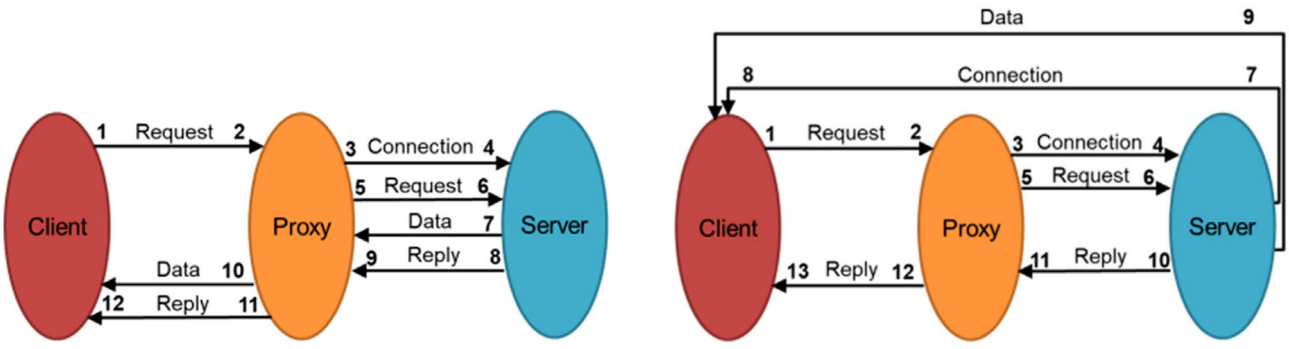
6 Related work

6.1 One-sided verbs versus two-sided verbs

Different design choices among one-sided verbs and two-sided verbs have been explored in several works [5, 8–10]. FaRM [5] exploits one-sided RDMA verbs to build a distributed shared memory system with transaction support. HERD [8] used a hybrid of unreliable one-sided verbs and two-sided verbs to keep fewer network round trip for all requests, avoiding RDMA READ operations. FaSST [9] chose to use two-sided verbs rather than one-sided verbs for a fast scalable all-to-all RPC system, assuming that packet drops are extremely rare. This design helps reduce the software complexity and increase scalability. Maomeng et al. [10] introduce a new RDMA paradigm called Remote Fetching Paradigm (RFP), where a client uses RDMA WRITE to send a request to a server and RDMA READ to fetch the result from a server while the request processing is done in server-side. RFP can be faster than a totally server-bypass model because it avoids "bypass access amplification" problem. Anuj et al. [11] present a guideline for building RDMA-based systems, which can be applied to our work to further optimize. SoftDC does not limit the type of RDMA verbs, allowing both one-sided verbs and two-sided verbs. The designs proposed in above works can be implemented with SoftDC.

6.2 Hybrid transport

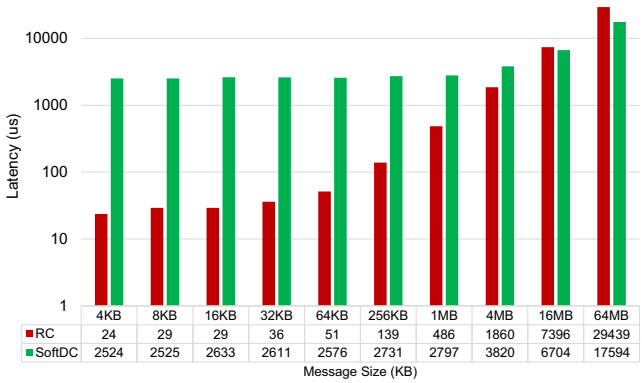
Multi-transport design using RC transport and UD transport has already been proposed in some works [2, 3]. MVAPICH-Aptus [2] is a multi-transport MPI design where there exist multiple communication channels of RCs and UDs and one of the channels is selected when sending a message according to the defined rules. Jithin et al. [3] present a hybrid transport model, which takes both strengths of RC and UD transport, and layer Memcached



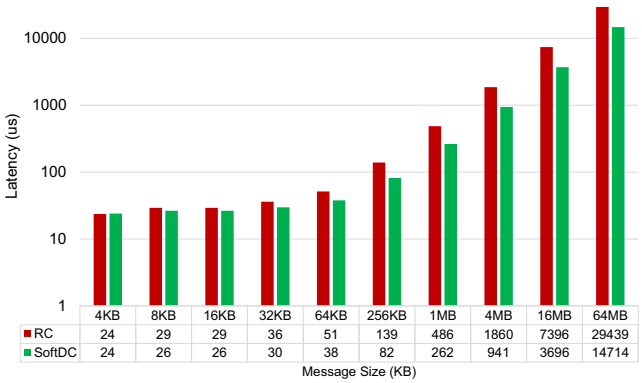
(a) RC QP-based Proxy System

(b) SoftDC-based Proxy System

Fig. 6 Latency of proxy system benchmark



(a) 1st Message



(b) 2nd Message

Fig. 7 Latency of micro-benchmark with proxy server

[12] on top of the hybrid transport. It allows active switching between RC and UD transports transparently to users, limiting the maximum number of RC connections. Above two works are different from our work since UD and RC have different roles to play in SoftDC: connection handling and data transfer respectively. Furthermore, their works focus on the design of RDMA application using RC and UD transports whereas SoftDC is an RDMA transport itself like RC transport.

6.3 Hardware-based solutions

There are some works that require hardware support. DrTM [13] exploits advanced hardware features such as RDMA and a hardware transactional memory (HTM) to build an order of magnitude faster transactional system than state-of-the-art systems. In contrast, our work does not rely on hardware support. Hari et al. [14] shares their experience with designing an MPI library using Dynamically Connected (DC) Infiniband transport. Although their work is based on the hardware implementation of DC transport, we believe that their MPI library can be easily rebuilt on top of our SoftDC transport with only a few changes.

7 Conclusion

This paper presents SoftDC, a totally software-based DC transport for any commodity Ethernet NIC. SoftDC utilizes the existing RDMA primitives such as RC QP and UD QP to provide the high functionality and performance of RC transport and the scalability of UD transport at the same time. Using the various benchmarks, we show that the software implementation of DC transport can deliver the similar performance to the hardware one in some cases. As shown in the example of NFS, we also demonstrate that SoftDC can be easily applicable to the existing RDMA system without many efforts. We believe that SoftDC has a potential to be the default transport for RDMA applications, replacing RC transport.

For future work, we plan to use UD QP for small messages whose size is less than MTU as well as connection messages and RC QP for large messages. This optimization would improve the performance of small message transfer. We will also investigate the feasibility of SoftDC in the existing MPI implementations.

Acknowledgements This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (2015M3C4A7065646, NRF-2017R1A2B4005681).

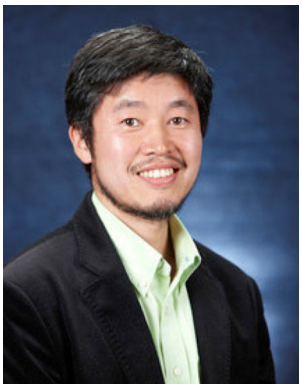
References

1. Park, J., Yeom, H.Y.: Design and implementation of software-based dynamically connected transport. In: International Workshop on Autonomic Management of High Performance Grid and Cloud Computing (AMGCC 2018), (2018)
2. Koop, M.J., Jones, T., Panda, D.K.: MVAPICH-Aptus: scalable high-performance multi-transport MPI over InfiniBand. In: IEEE International Symposium on Parallel and Distributed Processing, 2008, pp. 1–12. IPDPS 2008. IEEE (2008)
3. Jose, J., Subramoni, H., Kandalla, K., Wasi-ur-Rahman, M., Wang, H., Narravula, S., Panda, D. K.: Scalable memcached design for infiniband clusters using hybrid transports. In: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), pp. 236–243. IEEE Computer Society (2012)
4. MVAPICH2-X: Unified MPI+PGAS Communication Runtime over OpenFabrics/Gen2 for Exascale Systems, http://mvapich.cse.ohio-state.edu/
5. Dragojevic, A., Narayanan, D., Hodson, O., Castro, M.: FaRM: Fast remote memory. In: Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, pp. 401–414 (2014)
6. Haynes, T.: Network File System (NFS) Version 4 Minor Version 2 Protocol (2016)

7. Axboe, J.: Fio-flexible i/o tester synthetic benchmark. https://github.com/axboe/fio (2005). Accessed 28 Dec 2018
8. Kalia, A., Kaminsky, M., Andersen, D.G.: Using RDMA efficiently for key-value services. In: ACM SIGCOMM Computer Communication Review, vol. 44, pp. 295–306. ACM (2014)
9. Kalia, A., Kaminsky, M., Andersen, D.G.: FaSST: fast, scalable and simple distributed transactions with two-sided (RDMA) Datagram RPCs. In: OSDI, vol. 16, pp. 185–201 (2016)
10. Su, M., Zhang, M., Chen, K., Guo, Z., Wu, Y.: Rfp: When rpc is faster than server-bypass with rdma. In: Proceedings of the 12th European Conference on Computer Systems, pp. 1–15. ACM (2017)
11. Kalia, A., Kaminsky, M., Andersen, D.G.: Design guidelines for high performance RDMA systems. In: 2016 USENIX Annual Technical Conference, p. 437 (2016)
12. Memcached: A distributed Memory Object Caching System. https://memcached.org
13. Wei, X., Shi, J., Chen, Y., Chen, R., Chen, H.: Fast in-memory transaction processing using RDMA and HTM. In: Proceedings of the 25th Symposium on Operating Systems Principles, pp. 87–104. ACM (2015)
14. Subramoni, H., Hamidouche, K., Venkatesh, A., Chakraborty, S., Panda, D.K.: Designing MPI library with dynamic connected transport (DCT) of InfiniBand: early experiences. In: International Supercomputing Conference, pp. 278–295. Springer, Cham (2014)

Jiwoong Park received his B.S. degree in Department of Computer Science and Engineering from Korea University in 2013. Currently, he is a Ph.D. student in Deparment of Computer Science and Engineering in Seoul National University. His research interests are Operating, Distributed, and Database Systems.

Yongseok Son received his B.S. degree in Information and Computer Engineering from Ajou University in 2010, and the M.S. and Ph.D. degrees in Department of Intelligent Convergence Systems and Electronic Engineering and Computer Science at Seoul National University in 2012 and 2018, respectively. He was a postdoctoral research associate in Electrical and Computer Engineering at University of Illinois at Urbana-Champaign. Currently, he is an assistant professor in School of Computer Science and Engineering, Chung-Ang University. His research interests are Operating, Distributed, and Database Systems.



Heon Young Yeom is a Professor with the School of Computer Science and Engineering, Seoul National University. He received B.S. degree in Computer Science from Seoul National University in 1984 and his M.S. and Ph.D. degrees in Computer Science from Texas A&M University in 1986 and 1992 respectively. From 1986 to 1990, he worked with Texas Transportation Institute as a Systems Analyst, and from 1992 to 1993, he was with Samsung Data Systems as a Research Scientist. He joined the Department of Computer Science, Seoul National University in 1993, where he currently teaches and researches on Distributed Systems and Transaction Processing.

Yoonhee Kim she is the professor of Computer Science Department at Sookmyung Women's University. She received her Bachelors degree from Sookmyung Women's University in 1991, her Master degree and Ph.D. from Syracuse University in 1996 and 2001, respectively. She was a Research Staff Member at the Electronics and Telecommunication Research Institute during 1991 and 1994. Before joining the faculty of Sookmyung Women's University in 2001, she was the faculty of Computer Engineering dept. at Rochester Institute of Technology in NY, USA. Her research interests span many aspects of runtime support and management in distributed computing systems. She is a member of IEEE and OGF, and she has served on variety of program committees, advisory boards, and editorial boards.