Proceedings of the International Conference on Parallel Processing (ICPP 2007), XiAn, China, September 2007.
Also available as Argonne National Laboratory preprint ANL/MCS-P1422-0507.

Advanced Flow-control Mechanisms for the Sockets Direct Protocol over InfiniBand*

P. Balaji[†] S. Bhagvat[‡] D. K. Panda[§] R. Thakur[†] W. Gropp[†]

[†]Math. And Comp. Science,
Argonne National Laboratory
{balaji, thakur, gropp}@mcs.anl.gov

[‡]Scalable Systems Group,
Dell Inc.
sitha_bhagvat@dell.com

[§]Comp. Science and Engg.,
Ohio State University
panda@cse.ohio-state.edu

Abstract

The Sockets Direct Protocol (SDP) is an industry standard to allow existing TCP/IP applications to be executed on high-speed networks such as InfiniBand (IB). Like many other high-speed networks, IB requires the receiver process to inform the network interface card (NIC), before the data arrives, about buffers in which incoming data has to be placed. To ensure that the receiver process is ready to receive data, the sender process typically performs flow-control on the data transmission. Existing designs of SDP flow-control are naive and do not take advantage of several interesting features provided by IB. Specifically, features such as RDMA are only used for performing zero-copy communication, although RDMA has more capabilities such as sender-side buffer management (where a sender process can manage SDP resources for the sender as well as the receiver). Similarly, IB also provides hardware flow-control capabilities that have not been studied in previous literature. In this paper, we utilize these capabilities to improve the SDP flow-control over IB using two designs: *RDMA-based flow-control* and *NIC-assisted RDMA-based flow-control*. We evaluate the designs using micro-benchmarks and real applications. Our evaluations reveal that these designs can improve the resource usage of SDP and consequently its performance by an order-of-magnitude in some cases. Moreover we can achieve 10-20% improvement in performance for various applications.

1 Introduction

The Sockets Direct Protocol (SDP) [2] is an industry standard to allow existing TCP/IP applications to be executed on high-speed networks such as InfiniBand (IB) [1] and iWARP [5]. It is designed to transparently improve the performance of such applications by utilizing the hardware features of these networks. There are several implementations of SDP/IB. The first implementation [7] utilized IB send-receive operations to transmit data using intermediate buffer copies while taking advantage of IB's hardware protocol stack. Later designs [14, 6] extended this to utilize remote direct memory access (RDMA) to allow for zero-copy message transfers. Each design has its pros and cons. The buffer copy mechanism performs data copies during communication, adding overhead especially for large messages. Zero-copy approaches perform on-the-fly registration of buffers with the network interface card (NIC) and synchronization between the sender and receiver, adding overhead especially for small and medium-sized messages. Thus, to maximize performance, SDP stacks utilize the buffer copy mechanism for small and medium messages (up to 32KB), and zero-copy mechanisms for large messages (greater than 32KB). In this paper, we deal only with the buffer copy mechanism used for small and medium messages.

While the buffer copy design takes advantage of IB's hardware protocol stack, it is naive in aspects such as flow-control. Like many other high-speed networks, IB requires the receiver process to inform the NIC, before data arrives, about buffers in which incoming data has to be placed. To ensure that the receiver NIC is ready to receive data, the sender process performs flow-control on data transmission. The existing design of SDP flow-control uses send-receive-based communication, with each process managing its local flow-control buffers. With the receiver managing its local buffers, however, the sender is not aware of the receiver's exact usage status and layout. Thus, the flow-control tends to be conservative resulting in underutilization of buffers and performance loss.

RDMA, however, has more capabilities than just zero-copy communication. For example, it offers *sender-side buffer management*. Since RDMA is completely handled by the sender process, it allows this process to have complete control of SDP resources, such as flow-control buffers, on both the sender and receiver side. Further, IB provides hardware flow-control capabilities that have not been addressed so far.

In this paper, we propose two novel designs to improve the flow-control and performance for small and medium messages in SDP. The first design, *RDMA-based flow-control*, uses RDMA to allow the sender to manage both the sender and the receiver buffers. This design, as we will see in the later sections, achieves better utilization of the SDP buffer resources and consequently better performance. However,

*This research is funded in part by DOE grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; by NSF grants #CNS-0403342 and #CNS-0509452; by an STTR subcontract from RNet Technologies; and by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy (contract DE-AC02-06CH11357).

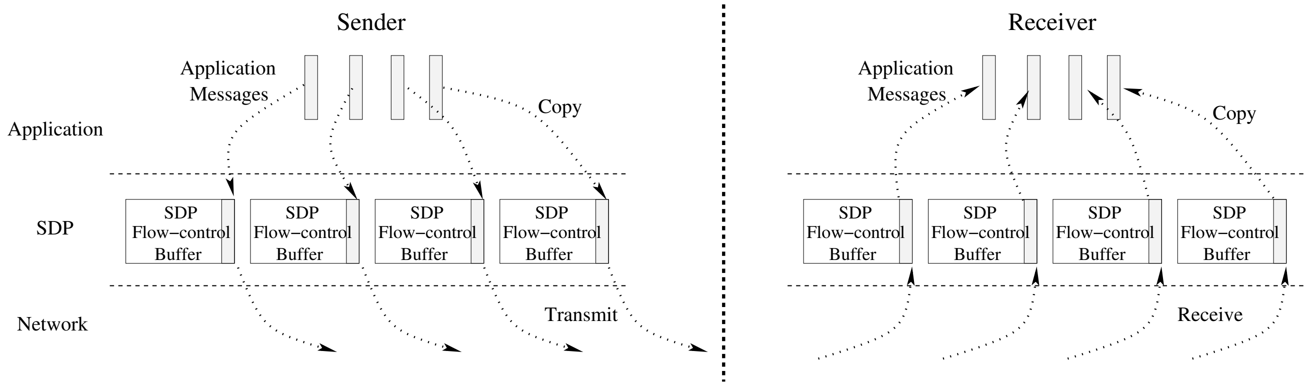

Figure 1: Credit-based Flow-control Mechanism

it assumes (from a performance standpoint) that the application will perform frequent communication to ensure that data is flushed out regularly from these buffers. Not doing so can result in performance penalty. The second design, *NIC-assisted RDMA-based flow-control*, utilizes IB's hardware flow-control to extend *RDMA-based flow-control* with asynchronous communication progress (i.e., data flushing) without sacrificing performance.

We demonstrate the capabilities of these designs using micro-benchmarks as well as real applications. Our results show that these designs can achieve almost an order-of-magnitude improvement in the bandwidth achieved by medium sized messages. Moreover, we can achieve performance improvements of about 10% in a virtual microscope application and close to 20% in an isosurface visual rendering application.

2 Existing Credit-based Flow-control

Several high-speed networks, including IB, require the receiver to prepost work queue entries (WQEs) informing the NIC about buffers in which messages should be received before each message arrives. To ensure that receive WQEs are posted before any data arrives, SDP performs flow-control. Currently, it uses a credit-based approach for achieving this. This flow-control is separate from IB's hardware flow-control and is a consequence of adopting existing designs of high-performance sockets on other networks [16, 15, 8].

2.1 Overview of Credit-based Flow-control

In credit-based flow-control (Figure 1), the sender is initially given a number of credits, say N. Each process allocates N SDP send and N SDP receive flow-control buffers, each of size S bytes. The receiver posts N receive WQEs to the NIC pointing to the receive flow-control buffers; that is, the next N messages will go into these buffers. On a `send()` call, each message smaller than S bytes is copied into a send buffer and transmitted to the corresponding receive buffer. Messages larger than S bytes are segmented and transmitted in a pipelined manner. On a `recv()` call, data is copied from the receive buffer to the destination buffer, and an acknowledgment (ACK) is sent to the sender informing it that the receive buffer is free to be reused. The sender loses a credit for every message sent and gains a credit for every ACK received.

Previous designs [7] also extend this design by delaying ACKs. In other words, the receiver sends an ACK after half the credits have been used instead of sending one after each received message. This approach reduces the amount of communication required and improves performance.

2.2 Limitations of Credit-based Flow-control

Credit-based flow-control has two primary disadvantages: buffer utilization and network utilization.

Buffer Utilization: In credit-based flow control, each message uses at least one credit irrespective of its size. Suppose the sender wants to send N 1B messages, and each SDP flow-control buffer is 8KB. Since the receiver has preposted N WQEs pointing to its receive buffers, each message is received in a separate buffer, effectively wasting 99.98% of the space allotted; in other words, only 1B of each 8KB SDP buffer is utilized. This wastage also reflects on the number of messages transmitted; excessive underutilization of buffer space results in the sender *believing* that it has used up the receiver resources, in spite of having free space available.

Network Utilization: In credit-based flow-control, on a `send()` call, SDP copies the message into the send flow-control buffer, waits until it has enough credits, and transmits the data to the receiver. Thus, when small and medium messages are transmitted, they are directly pushed to the network resulting in underutilization of the network and consequently performance loss. On the other hand, coalescing multiple small messages can allow SDP to transmit larger messages over the network and thus improve network utilization.

3 RDMA-based Flow-control

While credit-based flow-control is simple and widely accepted, it has several limitations, especially for small and medium messages. In this section, we describe RDMA-based

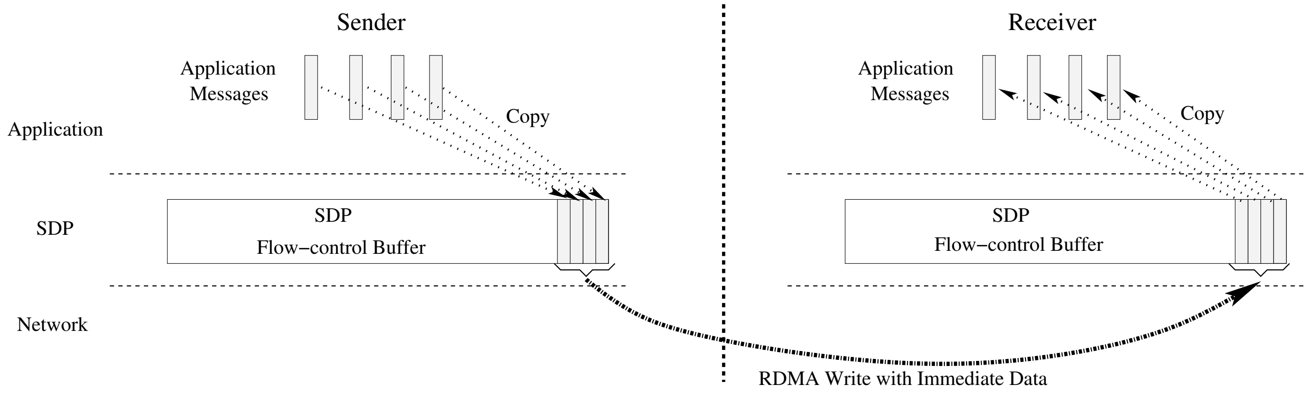

Figure 2: RDMA-based Flow-control Mechanism

flow-control, a new approach utilizing IB RDMA capability to improve the resource usage and performance of SDP.

3.1 Overview of RDMA-based Flow-control

Figure 2 illustrates RDMA-based flow-control, which differs from credit-based flow-control in two areas: improved buffer utilization and improved network utilization.

Improving Buffer Utilization: RDMA-based flow-control uses RDMA write with immediate data operations to allow the sender to manage where exactly data is buffered on the sender as well as the receiver SDP flow-control buffers. This approach allows data to be better packed, thus utilizing the buffers more efficiently. In credit-based flow-control, N SDP flow-control buffers each of size S are allocated, where N is the number of credits. In RDMA-based flow-control, on the other hand, one large flow-control buffer of size $(N \times S)$ is allocated. When the first message (size P) has to be communicated, it is placed (using RDMA write with immediate data) at the start of the receive buffer. When the second message of size Q has to be communicated, since the sender knows the exact usage of the receive buffer (the first P bytes are used), this message is written starting at byte $(P + 1)$ of the receiver buffer. This approach allows the sender to completely utilize the available space in the sender as well as receiver SDP buffers. On a `recv()` call, once data is copied from the receiver SDP buffer to the destination buffer, the receiver sends an acknowledgment to the sender informing it about the additional available space.

Improving Network Utilization: As long as space is available in the SDP receive buffer, RDMA-based flow-control follows a similar approach as credit-based flow-control; it sends out data before returning from the `send()` call. Once no more space is available on the receiver side, however, messages are copied into SDP send buffers, and control is returned to the application. This approach gives RDMA-based flow-control an opportunity to coalesce multiple small messages. When space is freed up in the SDP receive buffer, this data is sent as one large message instead of multiple small messages. This approach as two advantages. First, since as long as space is available in the receive buffer, data is sent out immediately, latency of small messages is not hurt. Second, when a large number of small or medium messages are transmitted, though the first few messages are sent immediately, the remaining are coalesced and sent as large messages, thus improving network utilization and performance.

In summary, RDMA-based flow-control avoids buffer wastage by using the RDMA's sender-side buffer management and improves network utilization and communication performance by coalescing messages.

3.2 Limitations of RDMA-based Flow-control

While RDMA-based flow-control can achieve better resource utilization and performance, it has one disadvantage: the lack of communication progress in some cases. Consider an example with an 64KB SDP flow-control buffer where the sender initiates 64 sends of 2KB each, total of 128KB. Of these, 32 messages (64KB) are directly transferred to the SDP buffer on the receiver. Then, if the receiver is not actively receiving data, the sender will run out of space in the receive buffer to write more data. Thus, the remaining 32 messages (64KB) are copied to the SDP send buffer, and control is returned to the application. At this time, suppose the sender goes into a large computation loop. The application on the receiver side, however, calls the `recv()` call, copies the 64KB it has already received, frees the SDP receive buffer, and sends an ACK to the sender informing it that the SDP receive buffer can be reused. In this situation, though the sender has buffered data to be sent and has been informed about available receiver buffer space, it cannot *see* this information until the application comes out of the computation loop and calls a communication function. Thus, communication progress is halted.

Note that credit-based flow-control does not face this limitation because for every `send()` call, if the sender does not have credits, it blocks until credits are received and posts the data to the network before returning control.

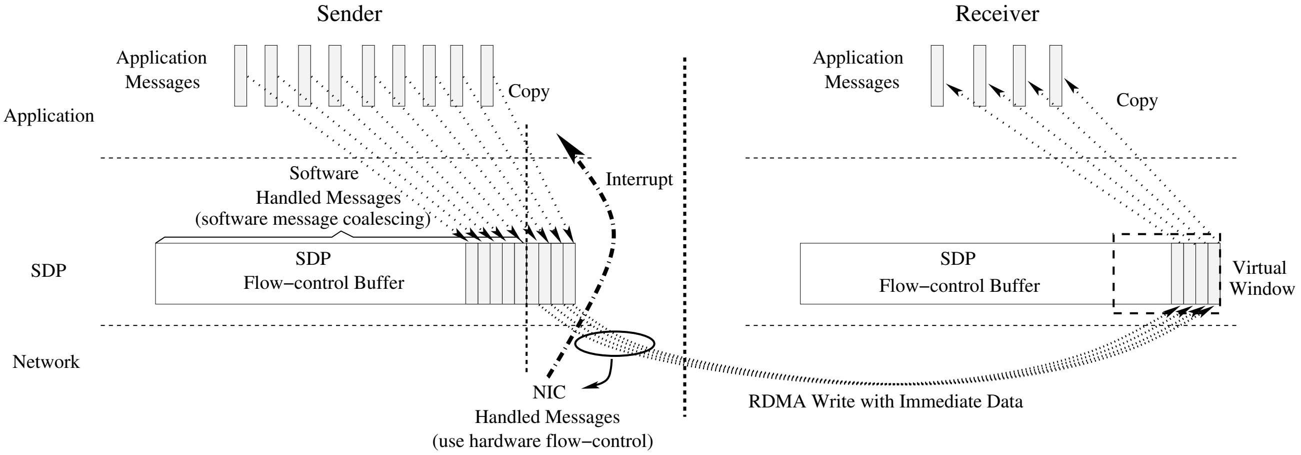

Figure 3: NIC-assisted RDMA-based Flow-control Mechanism

4 NIC-assisted Flow-control

Both credit-based flow-control and RDMA-based flow-control have disadvantages. Credit-based flow-control suffers from underutilization of SDP buffers and the network and results in low performance. While RDMA-based flow-control improves these aspects, it suffers from lack of communication progress when a large number of small messages have to be transmitted. To deal with these issues, we propose *NIC-assisted RDMA-based flow-control*. This mechanism extends RDMA-based flow-control by utilizing IB's hardware flow-control capabilities. In other words, it uses RDMA-based flow-control to coalesce messages as appropriate and improve performance and at the same time uses the IB hardware flow-control to ensure asynchronous communication progress.

NIC-assisted flow control comprises of two main sub-schemes: *virtual window scheme*, which aims at utilizing IB's hardware flow-control while handling its shortcomings, and *asynchronous interrupt scheme*, which enhances the virtual window scheme to improve performance by coalescing data.

4.1 Virtual Window Scheme

IB's hardware flow-control is not a byte-level flow-control, but rather a message-level flow-control; it makes sure that the sender NIC sends out only as many messages as the receiver NIC is expecting. The onus of ensuring that the receiver has appropriate buffer space for each message is on the upper layers such as SDP. To handle this situation, we utilize the *virtual window (W)* scheme. The primary idea of this scheme is to ensure that each posted receive WQE has a guarantee on the amount of buffer space available. For example, if the sender wants to send a message of 8KB, the receiver has to post a receive WQE only after 8KB of space is available.

In this scheme, the receiver posts a receive WQE only when at least the necessary virtual window size space is available in the SDP receive buffer. Thus, if the SDP buffer size is S bytes, the receiver initially posts S/W receive WQEs, where W is the virtual window size. The sender, likewise, makes sure that message segments posted to the network are always smaller than or equal to W bytes, by performing appropriate segmentation. Thus, the first S/W messages can definitely be accommodated in the SDP receive buffer. If the sender has to send more messages than S/W, it posts send WQEs corresponding to the additional data. However, since all the posted receive WQEs would be used up, IB hardware flow-control ensures that this data is not sent out by the sender NIC until the receiver posts additional receive WQEs.

We note that although each receive WQE corresponds to W bytes of available buffer space, this space can be anywhere in the SDP receive buffer; that is, the mapping between the WQE and the actual location of the corresponding buffer is not performed by the receiver. The sender uses RDMA write with immediate data operations to manage the actual buffer location to which each receive WQE maps. This flexibility allows the receiver to manage only the logical space allocated to each WQE, instead of the actual SDP buffer. For example, suppose the SDP buffer is 64KB and the virtual window is 8KB. The receiver initially posts 8 receive WQEs. The virtual window allocated to each receive WQE would be bytes (1 to 8K), (8K+1 to 16K), and so forth. Now, suppose the first message is only 1KB. In this case, the virtual windows corresponding to the remaining WQEs automatically shift by 7KB and would be bytes (1K+1 to 9K), (9K+1 to 17K), and so forth. The final 7KB is retained as free space. Since the sender is managing the actual SDP receive buffers, this shifting of the virtual windows is transparent to the receiver process. Later, if the second message that arrives is also 1KB, the virtual windows for the remaining WQEs again automatically shift and leave a total of 14KB of free space. Since this free space is more than the virtual window size (8KB), SDP can post an additional WQE, after which 6KB of free space will be available. When the receiver applications calls a `recv()`, the data in the SDP receive buffer is copied to the destination buffer, and more free space is created.

4.2 Asynchronous Interrupt Scheme

While the virtual window scheme provides capabilities to utilize IB hardware flow-control, it does not coalesce messages to improve performance. The asynchronous interrupt scheme is designed based on two primary goals: (i) coalesce messages to improve performance; (ii) utilize the virtual window scheme with IB hardware interrupts to carry out asynchronous communication progress without hurting performance.

Message Coalescing: In this scheme the SDP send buffer is divided into two portions: NIC-handled buffer and software-handled buffer (Figure 3). The NIC-handled buffer follows a similar pattern as the virtual window scheme. That is, data is copied into the SDP send buffer and a corresponding send WQE is posted to the NIC. The NIC uses hardware flow-control to send the data only after the receiver posts a receive WQE. After the NIC-handled buffer is full, data is copied into the software-handled buffer. This data is not directly sent out but is held, allowing it to be coalesced with later messages.

Asynchronous Communication Progress: During message coalescing, data is copied into the software-handled SDP buffer and control returned to the application. If more messages are communicated later, they can be coalesced together with this data to form larger messages and thus improve performance. If no other messages are communicated later, however, we need to asynchronously flush this data out. To do so, we request IB hardware interrupts for the messages in the NIC-handled buffer. Thus, once the first message that is queued in the NIC-handled buffer is transmitted, an interrupt is generated that is appropriately handled to flush out the data in the software-handled buffer as well. Although hardware interrupts are typically expensive, in this design the NIC can continue to transmit other messages in the NIC-handled buffer (using IB hardware flow-control), thus parallelizing the interrupt processing with communication. This design allows us to handle the interrupt without facing any performance penalty.

5 Experimental Results

In this section, we compare the performance of RDMA-based flow-control and NIC-assisted RDMA-based flow-control, with that of credit-based flow-control. We first describe the experimental test-bed in Section 5.1. Next, we evaluate the designs based on micro-benchmarks in Section 5.2 and then on real applications in Section 5.3.

5.1 Experimental Test-bed

The experimental test-bed consists of a 16-node cluster with dual 3.6 GHz Intel Xeon EM64T processors. Each node has a 2 MB L2 cache and 512 MB of 333 MHz DDR SDRAM. The nodes are equipped with Mellanox MT25208 InfiniHost III DDR PCI-Express adapters and are connected to a Mellanox MTS-2400, 24-port fully nonblocking DDR switch. The SDP stack is an in-house implementation at the Ohio State University. This stack is similar to other SDP stacks such as that available in the OpenFabrics distribution [4] except that it is completely in user-space and is built over the VAPI verbs interface provided by Mellanox Technologies.

For each experiment, ten or more runs/executions are conducted, the highest and lowest values are dropped (to discard anomalies), and the average of the remaining values is reported. For micro-benchmark evaluations, the results of each run are an average of 10,000 or more iterations.

5.2 Micro-benchmark Based Evaluation

In this section, we evaluate the flow-control designs using various micro-benchmark tests.

5.2.1 Ping-pong Latency and Uni-directional Bandwidth

Ping-pong Latency: Figure 4(a) shows the ping-pong latency of SDP with the three flow-control designs. In this experiment, the sender sends a message of size *S* to the receiver, on receiving which the receiver sends back another message of the same size to the sender. This is repeated several times and the total time averaged over the number of iterations to give the average round-trip time. The ping-pong latency reported here is one-half of the round-trip time, that is, the time taken for a message to be transferred from one node to another.

As shown in the figure, all three schemes perform identically. This result is expected as the three schemes differ only in the way they handle flow-control when there is either no remote credit available (credit-based flow-control) or no space available in the remote SDP buffer (RDMA-based and NIC-assisted flow-control). In the ping-pong latency test, only one message is communicated before the sender waits for a response from the remote process. Thus, there is no flow-control issue in this test and all schemes behave identically.

Unidirectional Bandwidth: Figure 4(b) shows the unidirectional bandwidth of the three flow-control mechanisms. In this experiment, the sender sends a single message of size *S* a number of times to the receiver. On receiving all the messages, the receiver sends back one small message to the sender indicating that it has received the messages. The sender calculates the total time, subtracts the one way latency of the message sent by the receiver, and based on the remaining time calculates the amount of data it had transmitted per unit time.

As shown in the figure, RDMA-based flow-control achieves the best performance, while credit-based flow-control achieves the worst, especially for small and medium-sized messages. For messages in the 256B to 4KB range, we notice almost an order of magnitude better performance. This behavior is expected because RDMA-based flow-control coalesces messages and thus utilizes the network more effectively resulting in a significantly better performance. In the figure, we also notice that the performance of NIC-assisted RDMA-based flow-control is very close to that of RDMA-based flow-control. This result shows that our scheme is able

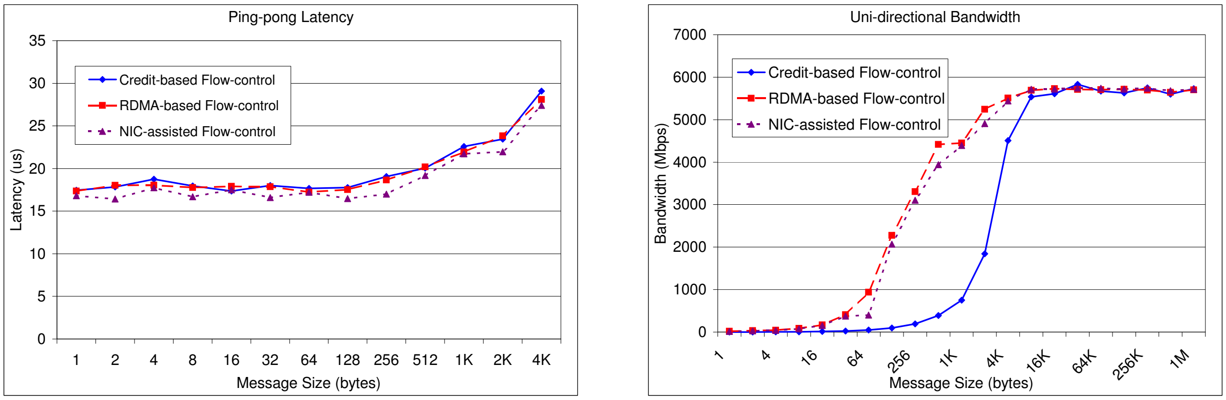

Figure 4: SDP micro-benchmark evaluation: (a) Ping-pong Latency and (b) Uni-directional Bandwidth

to effectively hide the cost of interrupt handling by overlapping interrupt processing with data transfer time.

5.2.2 Communication Progress Benchmark

The communication progress test is similar to a ping-pong latency test but with two changes. First, instead of one message being sent in each direction, a burst of 100 messages is used. Second, after each burst, an additional computation is added. If the flow-control scheme can achieve good communication progress, it can send out data even when the application is performing other computation. Thus, the receiver can receive the data immediately, and the computation on both the sender and the receiver is parallelized to some extent. However, if the flow-control scheme buffers data in its send buffer without performing good communication progress, the transmission of data is delayed until the computation is completed; that is, the receiver would be waiting to receive more data, which is available in the sender's SDP buffer but has not been transmitted. Only after the sender's computation is complete, when it tries to receive data, is this data flushed out. Thus, in this case, the computation on the sender and receiver is completely serialized resulting in poor performance.

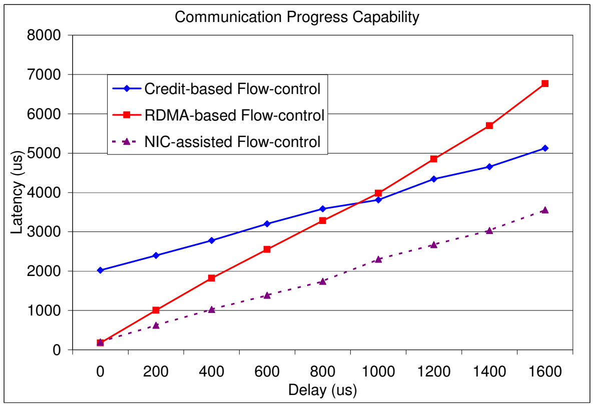

Figure 5: Communication Progress Benchmark

In Figure 5, we report the performance of the three flow-control schemes for a message size of 4KB with varying amounts of computation. In the figure, we notice that when there is no or minimal computation, RDMA-based flow-control and NIC-assisted RDMA-based flow-control take the least amount of time. Credit-based flow-control, on the other hand, takes the most time. As the amount of computation increases, however, we see that credit-based flow-control and NIC-assisted RDMA-based flow-control scale well, while RDMA-based flow-control deteriorates rapidly. In fact, for computation amounts greater than 1000μs, it is outperformed even by credit-based flow-control.

This test shows that credit-based flow-control and NIC-assisted flow-control are able to achieve good communication progress even when the application performs interleaving computation. For credit-based flow-control, when no remote credits are available, the scheme just blocks, waiting for the credits. Thus, the `send()` call does not return until the data is actually sent out. Consequently, the communication progress is good. For NIC-assisted RDMA-based flow-control, although data is buffered in the SDP send buffer without being immediately transmitted, the NIC interrupt ensures that the data is flushed out even when the application is busy with its computation. Thus, again the communication progress is good. RDMA-based flow-control, on the other hand, is not able to achieve good communication progress because this scheme buffers data hoping to coalesce it with later messages. Without communicating more messages, however, when the application starts doing additional computation, the buffered data has to wait without being flushed out.

5.2.3 Buffer Utilization Test

The buffer utilization test demonstrates the amount of SDP buffer space that is utilized by the different schemes. In this benchmark, we profile the SDP library to periodically monitor the amount of buffer space in which data is already copied and is not free to be used. The average percentage usage of the buffer space is measured and shown in Figures 6(a) (for 64KB SDP buffer size) and 6(b) (for 256KB SDP buffer size). We note two important aspects in these figures:

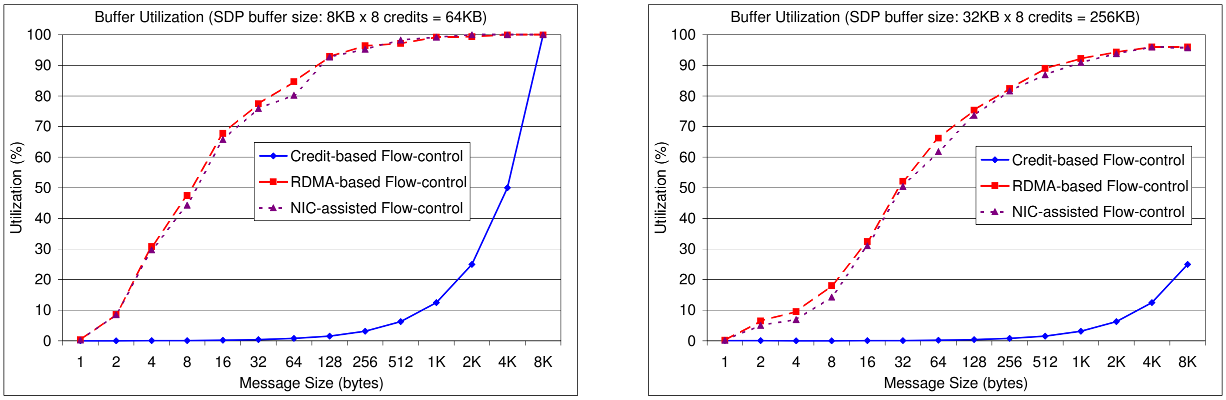

Figure 6: Buffer Utilization with SDP buffer size of : (a) 8KB x 8 credits = 64KB and (b) 32KB x 8 credits = 256KB

1. The buffer utilization of the RDMA-based flow-control and NIC-assisted RDMA-based flow-control is much higher than that of credit-based flow-control. This is attributed to the sender-side buffer management capability of RDMA, which allows data messages to be placed more compactly, thus allowing for improved buffer usage. In credit-based flow-control, when each SDP buffer is 8KB (Figure 6(a)), the scheme is able to reach 100% utilization only for message sizes of 8KB or higher. When each SDP buffer is 32KB (Figure 6(b)), the scheme achieves a maximum of 25% utilization.

2. Although the overall trend of these results is similar to the bandwidth test (Figure 4(b)), we notice that the buffer utilization peaks a lot more rapidly; that is, for a SDP buffer size of 64KB, peak buffer utilization is achieved at a message size of 512B itself. This indicates that the sender is able to pack data into the send buffers and is ready to transmit it, but the receiver is not able to receive data as fast, resulting in more data being accumulated in the SDP buffers and consequently a high utilization.

5.3 Application-based Evaluation

In this section, we evaluate the three flow-control designs based on two different applications, virtual microscope [12] and iso-surface visual rendering [11], that have been developed using the data-cutter library [9].

Overview of the Data-cutter Library: Data-Cutter is a component-based framework [10] developed by University of Maryland. It provides a framework, called filter-stream programming, for developing data-intensive applications. In this framework, the application processing structure is implemented as a set of components, called *filters*. Data exchange between filters is performed through a *stream* abstraction that denotes a unidirectional data flow from one filter to another. The overall processing structure of an application is realized by a *filter group*, which is a set of filters connected through logical streams. An application query is handled as a *unit of work* (UOW) by the filter group. The size of the UOW also represents the granularity in which data segments are distributed in the system and the granularity in which data processing is pipelined. Several data-intensive applications have been designed and developed by using the data-cutter run-time framework such as the virtual microscope application and the iso-surface visual rendering application.

Virtual Microscope: Virtual microscope [12] is a digitized microscopy application. The software support required to store, retrieve, and process digitized slides to provide interactive response times for the standard behavior of a physical microscope is a challenging issue [3, 12]. The main difficulty stems from the handling of large volumes of image data, which can range from a few hundreds of megabytes to several gigabytes. At a basic level, the software system should emulate the use of a physical microscope, including continuously moving the stage and changing magnification. The processing of client queries requires projecting high-resolution data onto a grid of suitable resolution and appropriately composing pixels mapping onto a single grid point.

Iso-surface Visual Rendering: Iso-surface rendering [13] is widely used technique in many areas, including environmental simulations, biomedical images, and oil reservoir simulators, for extracting and simplifying visualization of large datasets within a 3D volume. In this paper, we utilize a component-based implementation of such rendering [11].

Evaluation of the Data-cutter Applications: Figure 7 shows the performance of the virtual microscope and iso-surface visual rendering applications for the different flow-control designs. Both applications have been executed with a UOW of 1KB. The complete dataset is about 1GB, which is hosted on a *RAM disk* in order to avoid disk fetch overheads in the experiment. The virtual microscope application used five filters: *read data*, *decompress*, *clip*, *zoom*, and *view*. For this application, five instances of the filter group (total 25 filters) were placed on 13 dual-processor nodes. The iso-surface visual rendering application used four filters: *read dataset*, *iso-surface extraction*, *shade and rasterize*, and *merge/view*. For this application, six instances of the filter group (total 24

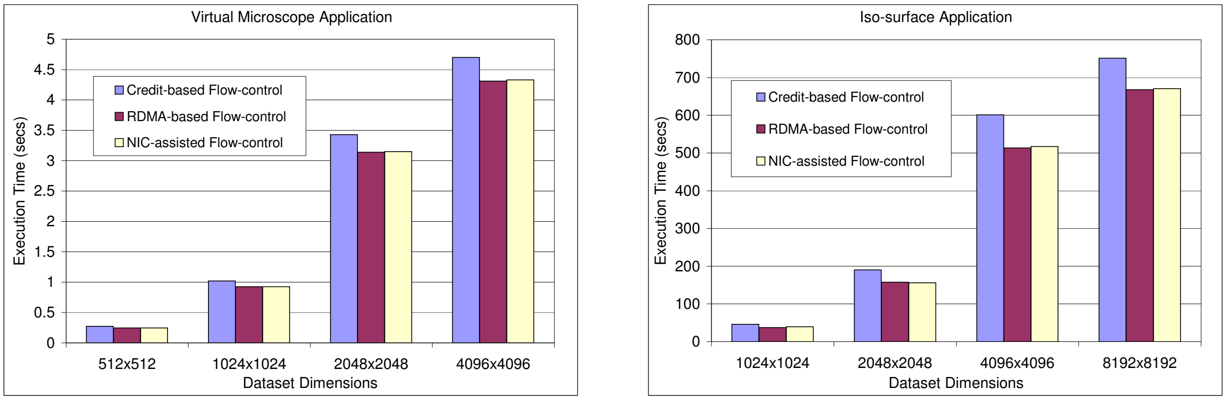

Figure 7: Application Evaluation with Data-cutter: (a) Virtual Microscope and (b) Iso-surface Visual Rendering

filters) were placed on 12 dual-processor nodes. Each filter performs some computation and communicates the processed data to the next filter. Once the communication is initiated, the filter starts computation on the next UOW, thus attempting to overlap communication with computation.

As shown in the figure, credit-based flow-control shows poor performance for both applications with all dataset sizes compared to RDMA-based and NIC-assisted RDMA-based flow-control. In these applications, since multiple UOWs are processed and communicated to the next filter, the coalescing capability of these designs allows them to utilize the network more effectively and hence achieve better performance. Our designs outperform credit-based flow-control by around 10% for the virtual microscope application and close to 20% for the iso-surface visual rendering application.

We also notice no difference in the performance of RDMA-based and NIC-assisted RDMA-based flow-control. This result shows that the enhanced communication progress is not very beneficial since the applications themselves frequently make communication calls to ensure such progress.

6 Conclusions

In this paper, we discussed the limitations of the existing credit-based flow-control in the Sockets Direct Protocol (SDP) over IB. We pointed out that SDP currently does not take advantage of the various features provided by IB. For example, RDMA is used only for zero-copy communication, and its other capabilities such as *sender-side buffer management* are unutilized. Similarly, IB's hardware flow-control has not been harnessed so far. We proposed two new flow-control mechanisms, known as RDMA-based flow-control and NIC-assisted RDMA-based flow-control, to handle these limitations and improve the resource usage and performance of SDP. We presented a detailed overview of the two designs and evaluated them using micro-benchmarks as well as applications. Our results show that these schemes can achieve nearly an order-of-magnitude improvement in the bandwidth achieved by SDP and around 10-20% improvement in application performance.

References

[1] InfiniBand Trade Association. http://www.infinibandta.com.

[2] SDP Specification. http://www.rdmaconsortium.org/home.

[3] A. Afework, M. D. Beynon, F. Bustamante, A. Demarzo, R. Ferreira, R. Miller, M. Silberman, J. Saltz, A. Sussman, and H. Tsang. Digital Dynamic Telepathology - The Virtual Microscope. In *AMIA*, 1998.

[4] OpenFabrics Alliance. http://www.openib.org.

[5] S. Bailey and T. Talpey. Remote Direct Data Placement (RDDP), April 2005.

[6] P. Balaji, S. Bhagvat, H.-W. Jin, and D. K. Panda. Asynchronous Zero-copy Communication for Synchronous Sockets in the Sockets Direct Protocol (SDP) over InfiniBand. In *CAC*, 2006.

[7] P. Balaji, S. Narravula, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Sockets Direct Protocol over InfiniBand in Clusters: Is it Beneficial? In *ISPASS*, 2004.

[8] P. Balaji, P. Shivam, P. Wyckoff, and D. K. Panda. High Performance User Level Sockets over Gigabit Ethernet. In *Cluster*, 2002.

[9] M. Beynon, T. Kurc, A. Sussman, and J. Saltz. Design of a Framework for Data-Intensive Wide-Area Applications. In *HCW*, 2000.

[10] M. D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz. Distributed Processing of Very Large Datasets with DataCutter. *Parallel Computing*, October 2001.

[11] M. D. Beynon, T. Kurc, U. Catalyurek, and J. Saltz. A Component-based Implementation of Iso-surface Rendering for Visualizing Large Datasets. *Report CS-TR-4249 and UMIACS-TR-2001-34, University of Maryland, Department of Computer Science and UMIACS*, 2001.

[12] U. Catalyurek, M. D. Beynon, C. Chang, T. Kurc, A. Sussman, and J. Saltz. The Virtual Microscope. *IEEE Transactions on Information Technology in Biomedicine*, 2002.

[13] J. Gao and H. Shen. Parallel View Dependent Isosurface Extraction using Multi-Pass Occlusion Culling. In *ACM/IEEE Symposium on Parallel and Large Data Visualization and Graphics*, 2001.

[14] D. Goldenberg, M. Kagan, R. Ravid, and M. Tsirkin. Zero Copy Sockets Direct Protocol over InfiniBand - Preliminary Implementation and Performance Analysis. In *HotI*, 2005.

[15] J. S. Kim, K. Kim, and S. I. Jung. SOVIA: A User-level Sockets Layer Over Virtual Interface Architecture. In *Cluster*, 2001.

[16] H. V. Shah, C. Pu, and R. S. Madukkarumukumana. High Performance Sockets and RPC over Virtual Interface (VI) Architecture. In *CANPC*, 1999.