Designing and Enhancing the Sockets Direct Protocol (SDP) over iWARP and InfiniBand

A Thesis

Presented in Partial Fulfillment of the Requirements for

the Degree Master of Science in the

Graduate School of The Ohio State University

By

Sitha Bhagvat, B. E.

* * * * *

The Ohio State University

2006

Master's Examination Committee:

Prof. Dhabaleswar K. Panda, Adviser

Prof. P. Sadayappan

Approved by

____

Adviser

Graduate Program in
Computer Science and
Engineering

ABSTRACT

Earlier generation protocols such as TCP/IP have traditionally been implemented in the host kernel space and have not been able to scale with the increasing network speeds. Accordingly, they form the primary communication bottleneck in current high-speed networks. In order to allow existing TCP/IP applications that had been written on top of the sockets interface to take advantage of high-speed networks, researchers have come up with a number of solutions including high-performance sockets. The primary idea of high-performance sockets is to build a pseudo sockets-like implementation which utilizes the advanced features of high-speed networks while maintaining the TCP/IP sockets interface; this allows existing TCP/IP sockets based applications to transparently achieve a high performance. The Sockets Direct Protocol (SDP) is an industry standard for such high-performance sockets over the InfiniBand (IB) network and the Internet Wide-Area RDMA Protocol (iWARP) over Ethernet networks.

In this thesis, we focus on designing and enhancing SDP over iWARP and IB. Specifically, we divide the research performed into two parts: (ii) designing SDP over iWARP, understanding the various implications associated with such a design and proposing a framework integrating SDP and iWARP to allow for higher performance and (ii) designing various enhancements for the SDP implementation over IB to better utilize the support provided by the network adapter and thus improve end performance. We also provide details of our designs in this thesis and present preliminary performance studies to indicate the promise offered by these designs.

This thesis is dedicated to my parents.

ACKNOWLEDGMENTS

I would like to thank my adviser, Prof. D. K. Panda for his guidance and support through the course of my graduate studies. I appreciate the time and effort he invested in steering my research work. I am thankful to Prof. P. Sadayappan for agreeing to serve on my Master's examination committee.

My thanks are due to Pavan Balaji for all his help and constant motivation that was imperative for the successful completion of my Masters program. I would also like to thank him for his valuable comments on this thesis.

I would like to thank Karthik and Jin for their immense help and encouragement. I also acknowledge the student members of NOWLAB for their willingness to help with everything, at all times. Special thanks to Tom, Ewana, Elizabeth, Catrena and Marty of the CSE department for making my stay at OSU a pleasant one.

Finally, I would like to thank my friends Pavan, Aditya, Harsha, Swapna, Supriya for all the support and encouragement.

I feel greatly indebted to my parents, sister, brother and my entire family for their love, affection and confidence in me.

VITA

March 28, 1983	Born - Hyderabad, India
Nov 2000- May 2004	B.E., Computer Science and Engineering, Osmania University, Hyderabad, India
Feb 2004 - June 2004	Intern, PennyWise Solutions, Hyderabad
Fall 2005 - Summer 2006	Graduate Research Associate, The Ohio State University.

PUBLICATIONS

Research Publications

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." *Workshop on Communication Architecture for Clusters (CAC); in conjunction with the International Parallel and Distributed Processing Symposium (IPDPS), Rhodes Island, Greece, April, 2006.*

FIELDS OF STUDY

Major Field: Computer Science and Engineering

TABLE OF CONTENTS

Page

Abstract ii

Dedication iv

Acknowledgments v

Vita vi

List of Figures ix

Chapters:

1. Introduction 1

    1.1 Sockets Direct Protocol: State-of-the-Art and Limitations 3
    1.2 Thesis Overview and Proposed Framework 5

2. Background and Motivation 8

    2.1 Overview of iWARP 8
        2.1.1 Remote Direct Data Placement (RDDP) 9
        2.1.2 Marker PDU Aligned (MPA) 10
    2.2 Overview of InfiniBand Architecture 12
        2.2.1 IB Communication 14
        2.2.2 RDMA Communication Model 14
        2.2.3 Atomic Operations Over IB 15
    2.3 Sockets Direct Protocol 16

3. Designing the Sockets Direct Protocol (SDP) over iWARP 18

3.1 Design Choices for iWARP 20
3.2 Sockets Direct Protocol over iWARP 21
3.2.1 Generic SDP/iWARP design 22
3.2.2 Design of Integrated SDP/iWARP Framework 24
3.3 Performance Results 26
3.3.1 Experimental testbed 26
3.3.2 iWARP Evaluation 26
3.3.3 SDP evaluation 29
3.4 Summary 33

4. Designing NIC-assisted Flow Control for the Sockets Direct Protocol (SDP) over InfiniBand 35

4.1 Overview of Existing Flow Control Mechanisms 37
4.1.1 Credit-based Flow-control in SDP 37
4.1.2 Limitations with Credit-based Flow Control 39
4.1.3 Packetized Flow control in SDP 39
4.1.4 Limitations with Packetized Flow Control 41
4.2 Overview of InfiniBand Message Level Flow control 42
4.3 Design of NIC-assisted Flow Control 43
4.3.1 Virtual Window-based Scheme 44
4.3.2 Asynchronous Interrupt-based Scheme 45
4.4 Experimental Results 47
4.5 Summary 52

5. Concluding Remarks and Future Work 53

5.1 Summary of Research Contributions 53
5.2 Future Research Directions 54

Bibliography 56

LIST OF FIGURES

Figure **Page**

1.1 Kernel-level and User-level Sockets Implementation on IB (a) Traditional TCP/IP Stack (b) SDP over InfiniBand 3

1.2 Proposed Framework: (a) SDP/iWARP (b) Flow-control in SDP/IB . 6

2.1 Marker PDU Aligned (MPA) protocol Segment format 12

2.2 InfiniBand Architecture (Courtesy InfiniBand Specifications) 13

3.1 (a) 10 Gigabit Ethernet TOE stack (b) iWARP protocol stack 19

3.2 Sockets Direct Protocol over iWARP 22

3.3 iWARP Latency 27

3.4 iWARP Bandwidth 27

3.5 SDP/iWARP (with markers) and SDP/iWARP (no markers): Latency 31

3.6 SDP/iWARP (with markers) and SDP/iWARP (no markers): Bandwidth 31

3.7 SDP Micro-benchmarks: Latency 32

3.8 SDP Micro-benchmarks: Bandwidth 33

4.1 The Credit Based Approach 38

4.2 Packetized Flow Control Approach 40

4.3 NIC-assisted Flow Control Approach 46

4.4 SDP/IB with NIC-assisted Flow Control Micro-benchmarks: Latency 48

4.5 SDP/IB with NIC-assisted Flow Control Micro-benchmarks: Bandwidth 49

4.6 (a) Good communication progress; computation loops are highly parallelized and (b) Bad communication progress; computation loops are not parallelized 49

4.7 SDP/IB with NIC-assisted Flow Control Micro-benchmarks: Computation Communication Overlap Test 50

CHAPTER 1

INTRODUCTION

Cluster systems are becoming increasingly popular in various application domains mainly due to their high performance-to-cost ratio. Out of the current Top 500 Supercomputers, 364 systems are clusters [26]. Cluster systems are now present at all levels of performance, due to the increasing capability of commodity processors, memory and the network communication stack. Since the nodes in a cluster system rely on the network communication stack in order to coordinate and communicate with each other, it forms a critical component in the efficiency and scalability of the system. Therefore, it is of particular interest. The network communication stack itself comprises of two components: (i) the network hardware and (ii) the communication protocol and the associated software stack.

With respect to the first component, during the last few years, the research and industry communities have been proposing and implementing high-speed networking hardware such as InfiniBand (IB) [4], 10-Gigabit Ethernet (10GigE) [20, 27, 28, 21] and Myrinet [15], in order to provide efficient support for such cluster systems amongst others. For the second component (communication protocol stack), however, there has not been as much success.

Earlier generation communication protocols such as TCP/IP [38, 42] relied upon the kernel for processing the messages. This caused multiple copies and kernel context switches in the critical message passing path. Thus, the communication performance was low. Researchers have been looking at alternatives to increase the communication performance delivered by clusters in the form of low-latency and high-bandwidth user-level protocols such as FM [33] and GM [18] for Myrinet [15], EMP [37, 36] for Gigabit Ethernet [23], etc.

These developments are reducing the gap between the performance capabilities of the physical network and that obtained by the end users. While this approach is good for developing new applications, it might not be so beneficial for existing applications. A number of applications have been developed on kernel-based protocols such as TCP/IP or UDP/IP using the sockets interface. To support such applications on high performance user-level protocols without any changes to the application itself, researchers have come up with different techniques including high-performance sockets implementations [12, 30, 35, 14]. High-performance sockets are pseudo sockets-like implementations to meet two primary goals: (i) to directly and transparently allow existing sockets applications to be deployed on to clusters connected with modern networks such as IB and iWARP and (ii) allow such deployment while retaining most of the raw performance provided by the networks.

In an attempt to standardize these efforts towards high-performance sockets implementations, the RDMA Consortium brought out a new standard known as the Sockets Direct Protocol (SDP) [2]. Figure 1.1 shows the traditional TCP/IP stack and the SDP stack over IB or iWARP.

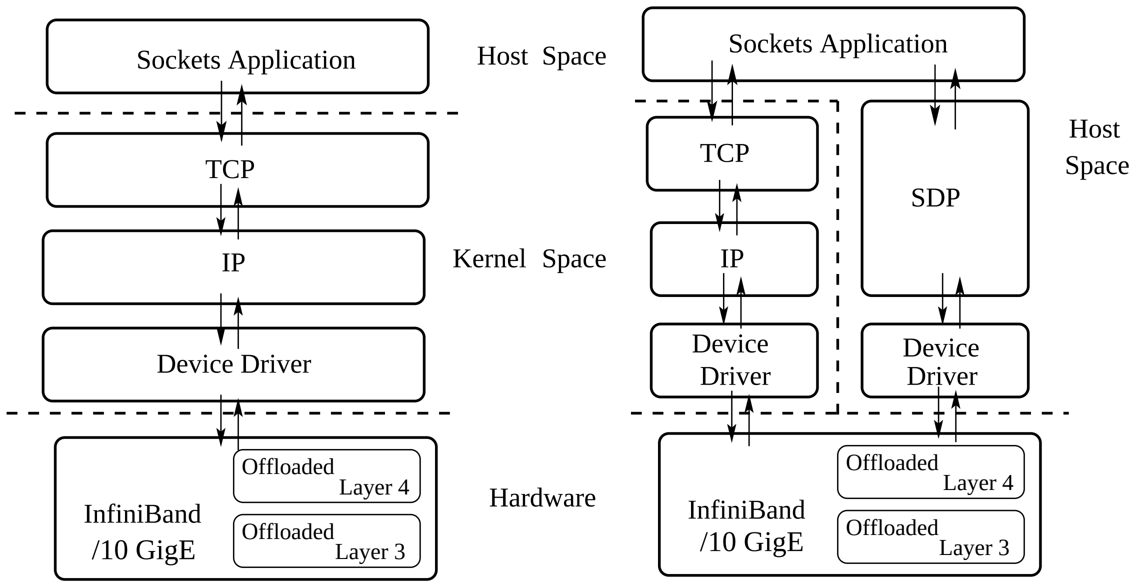

Figure 1.1: Kernel-level and User-level Sockets Implementation on IB (a) Traditional TCP/IP Stack (b) SDP over InfiniBand

1.1 Sockets Direct Protocol: State-of-the-Art and Limitations

As indicated earlier, the SDP standard attempts to transparently provide high performance for existing sockets-based applications over high-speed networking stacks such as IB and iWARP. While, there are several implementations of the SDP standard [10, 25, 24, 7], these lack in several aspects. Some of these aspects correspond to designs proposed in the SDP standard which might not be optimal in all scenarios, while the others are specific to existing SDP implementations where the current designs have scope for improvement in multiple dimensions. In this section, we point out three such limitations:

1. The SDP standard deals with IB and iWARP uniformly. This means that, the SDP implementation has to be independent of the underlying network communication stack. In reality, however, IB and iWARP are vastly different. IB has been designed mainly for SAN environments, whereas iWARP is aimed to bring the SAN capabilities (e.g., RDMA) to the WAN. The way each of these network stacks is implemented could affect the performance of SDP significantly.

2. Communication efficiency in any protocol stack largely depends on the efficiency of the flow-control mechanism; this dependency is typically higher for small messages where the actual data transfer time is not very large. The SDP standard specifies a flow control mechanism known as credit-based flow control for this. As we will see in the later chapters, the credit based flow control mechanism is quite inefficient in resource usage and suffers from poor performance. In order to deal with the limitations of credit-based flow-control, previous researchers have proposed other mechanisms such as the packetized flow-control [6]. While packetized flow control achieves a better resource usage and performance as compared to credit-based flow control it has other costs such as non-optimal communication progress.

3. For large message transfers, the SDP standard specifies a zero-copy based mechanism. However, this scheme has been proposed keeping the capabilities of a certain class of sockets interface known as the asynchronous sockets in mind. For the more widely used class of sockets interface, i.e., synchronous sockets, this scheme does not provide the best performance. We had previously

proposed a scheme known as Asynchronous Zero-copy SDP (AZ-SDP) [7] to handle this limitation.

In this thesis we will study the first two drawbacks of SDP.

1.2 Thesis Overview and Proposed Framework

Keeping in mind the issues described in Section 1.1, we propose the two following objectives:

1. Design and implement SDP over iWARP and understand the implications of such an implementation. Specifically, we analyze the limitations of a generic implementation of SDP over iWARP (i.e., an implementation which is decoupled with the iWARP design), and propose a framework to integrate SDP and iWARP to achieve high performance.

2. Design enhancements for SDP over IB to better utilize the support provided by the network adapter and thus improve end performance.

The proposed framework can be seen in the Figure 1.2. The major components of this framework are the SDP/iWARP and flow control aspects in SDP/IB.

Different designs for iWARP exist. Some of these achieve complete WAN compatibility while others choose to relax this constraint to achieve a better performance. As we will see in Chapter 3, if complete WAN compatibility is desired, traditional designs of SDP used in the context of iWARP do not provide the best performance. For the SDP/iWARP component, we take a two step approach: (i) we first analyze the failure of traditional SDP designs over completely WAN compatible iWARP and (ii) design a new framework integrating SDP and iWARP, which not

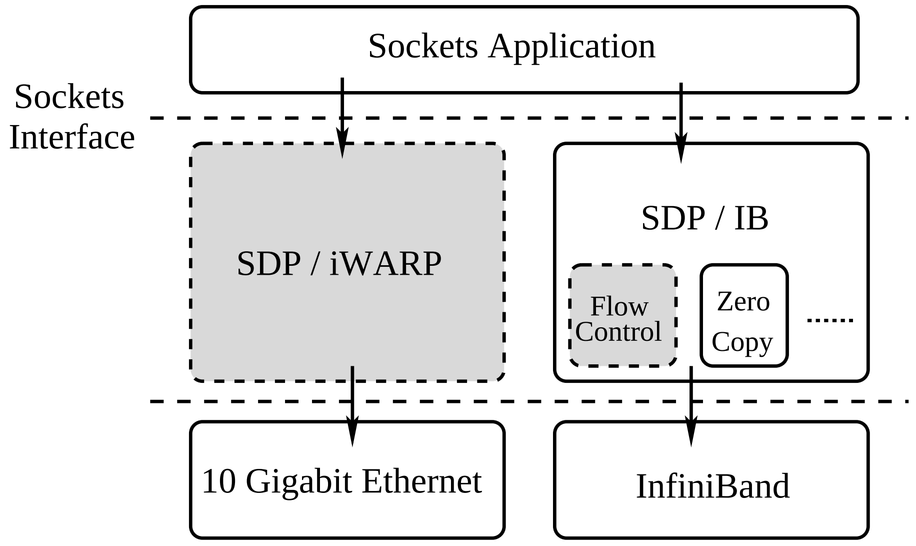

Figure 1.2: Proposed Framework: (a) SDP/iWARP (b) Flow-control in SDP/IB

only achieves complete WAN compatibility, but also does it while achieving a high performance.

For the flow-control in SDP/IB component, we design a new NIC-assisted flow-control mechanism for SDP/IB which retains the benefits of existing flow control mechanisms while efficiently handling their limitations. Specifically, it extends the previously proposed packetized flow-control mechanism to achieve its efficient resource utilization and high performance while handling its limitations with respect to communication progress.

The rest of this thesis is arranged as follows. We present detailed background of iWARP, InfiniBand, and SDP in Chapter 2. We present the detailed design of SDP over iWARP in Chapter 3, along with the analysis of the effects of such an implementation. In Chapter 4, we describe the issues with the existing SDP flow control mechanisms and propose a new flow control mechanism which utilizes the hardware capabilities of IB to address the limitations of existing mechanisms. Conclusions and future work are indicated in Chapter 5.

CHAPTER 2

BACKGROUND AND MOTIVATION

In this chapter, we start with an overview of iWARP in Section 2.1. Next, in Section 2.2, we provide a brief background on the InfiniBand (IB) architecture and a subset of its features. The Sockets Direct Protocol (SDP) is described in Section 2.3.

2.1 Overview of iWARP

Though hardware-based implementations of TCP/IP, such as the TCP Offload Engines (TOEs), have been able to handle most of the inefficiencies of the host-based TCP/IP stack, their capabilities are limited [11, 29, 31, 9, 13]. For example, TOEs do not provide capabilities such as Remote Direct Memory Access (RDMA) or zero-copy communication, which are common features in other high-speed interconnects (e.g., IB).

The Internet Wide Area RDMA Protocol (iWARP) is a new initiative by the Internet Engineering Task Force (IETF) [22] and the Remote Direct Memory Access Consortium (RDMAC) [17] in order to meet these limitations. The iWARP standard, when offloaded on to the network adapter, provides two primary extensions to the TOE stack: (i) it exposes a rich interface including zero-copy, asynchronous and one-sided communication primitives and (ii) it extends the TCP/IP implementation on the TOE to allow such communication while maintaining compatibility with the existing TCP/IP implementations.

The iWARP stack comprises of up to three protocol layers on top of a reliable IP-based protocol such as TCP: (i) RDMA Protocol (RDMAP) interface, (ii) Remote Direct Data Placement (RDDP) [5] layer and (iii) Marker PDU Aligned (MPA) [19] layer.

The RDMAP layer is a thin interface which allows applications to interact with the RDDP layer. The RDDP layer uses an IP based reliable protocol stack such as TCP to perform the actual data transmission. The MPA stack is an extension to the TCP/IP stack in order to maintain backward compatibility with the existing infrastructure. Details about the RDDP and MPA layers are provided in Sections 2.1.1 and 2.1.2, respectively.

2.1.1 Remote Direct Data Placement (RDDP)

The iWARP standard was developed to serve two purposes. First, the protocol should be able to provide high performance in System-area Network (SAN) environments by utilizing an offloaded protocol stack and zero-copy data transfer between host memories. Second, the protocol should maintain compatibility with the existing IP infrastructure using an implementation over an IP based reliable transport layer stack. Maintaining these two features involves novel designs for several aspects. We describe some of these designs corresponding to the RDDP layer, in this section.

In-Order Delivery and Out-of-Order Placement: RDDP relies on decoupling of placement and delivery of messages, i.e., placing the data in the user buffer is performed in a decoupled manner with informing the application that the data has been placed in its buffer. In this approach, the sender breaks the message into multiple segments of MTU size; the receiver places each segment directly into the user buffer, performs book-keeping to keep track of the data that has already been placed and once all the data has been placed, informs the user about the arrival of the data. This approach has two benefits: (i) there are no additional copies for unexpected messages involved in this approach and (ii) when a segment is dropped, the future segments do not need to be buffered till this segment arrives; they can directly be placed into the user buffer as and when they arrive. This approach, however, involves two important features to be satisfied by each segment: *self-describing* and *self-contained* segments. The *self-describing* property of segments involves adding enough information in the segment header so that each segment can individually be placed at the appropriate location without any information from the other segments. The information contained in the segment includes the message sequence number (MSN), the offset in the message buffer to which the segment has to be placed (MO) and others. *Self-containment* of segments involves making sure that each segment contains either a part of a single message, or the whole of a number of messages, but not parts of more than one message.

2.1.2 Marker PDU Aligned (MPA)

The RDDP protocol by itself has several deficiencies as listed below:

1. RDDP is an end-node protocol. The intermediate nodes do not have to support RDDP. This leads to a problem known as "Middle Box Fragmentation"

for Layer-4 switches. Layer-4 switches (e.g., firewalls, load-balances) are transport protocol specific and capable of making decisions regarding the forwarding of the arriving message segments. In order to optimize such forwarding, several implementations of layer-4 switches support techniques such as TCP Splicing [16] which relax the requirement for one-to-one correspondence between the segments coming in and the segments going out of the switch, i.e., the segments coming in might be re-fragmented and/or re-assembled at the switch. Thus, if the network infrastructure has layer-4 switches in the path (e.g., Internet traffic), it is practically impossible for the end node to recognize the RDDP headers especially when some packets are dropped. This mandates that the protocol not assume the self-containment property at the receiver end, and add additional information in each segment to help recognize the RDDP header.

2. It has been previously shown that the data-integrity check performed by TCP/IP (i.e., checksum) is not entirely reliable in several environments [39]. Accordingly, end users have demanded additional stronger data-integrity checks to ensure no data corruption. Several upper layers such as iSCSI [34] typically perform additional data integrity checks such as the 32-bit Cyclic Redundancy Check (CRC-32) for this reason.

In order to tackle this problem, iWARP introduces a new protocol layer known as MPA. Figure 2.1 illustrates the new segment format with MPA. This new segment is known as the Framing Protocol Data Unit (FPDU). The FPDU format has three essential changes:

- Markers: Strips of data to pointing to the RDDP header. These are spaced uniformly based on the TCP sequence number and provide the receiver with a deterministic way to find them and eventually the right header for the segment.

- Cyclic Redundancy Check (CRC): A stronger data integrity check.

- Segment Pad Bytes: Pad bytes to ensure alignment with the TCP segment.

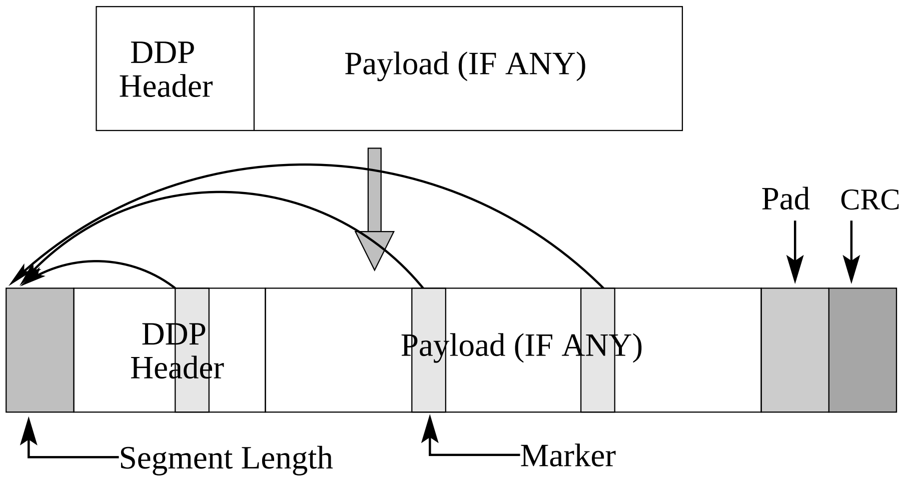

Figure 2.1: Marker PDU Aligned (MPA) protocol Segment format

2.2 Overview of InfiniBand Architecture

The InfiniBand Architecture (IB) is an industry standard that defines a System Area Network (SAN) to design clusters offering low latency and high bandwidth. The compute nodes are connected to the IB fabric by means of Host Channel Adapters (HCAs). IB defines a semantic interface called as Verbs for the consumer applications to communicate with the HCAs. VAPI is one such interface developed by Mellanox Technologies [1]. Other such Verbs interfaces (e.g., Gen2 verbs) also exist.

IB mainly aims at reducing the system processing overhead by decreasing the number of copies associated with a message transfer and removing the kernel from the critical message passing path. This is achieved by providing the consumer applications direct and protected access to the HCA. The specification for the verbs interface includes a queue-based interface, known as a Queue Pair (QP), to issue requests to the HCA. Figure 2.2 illustrates the InfiniBand Architecture model.

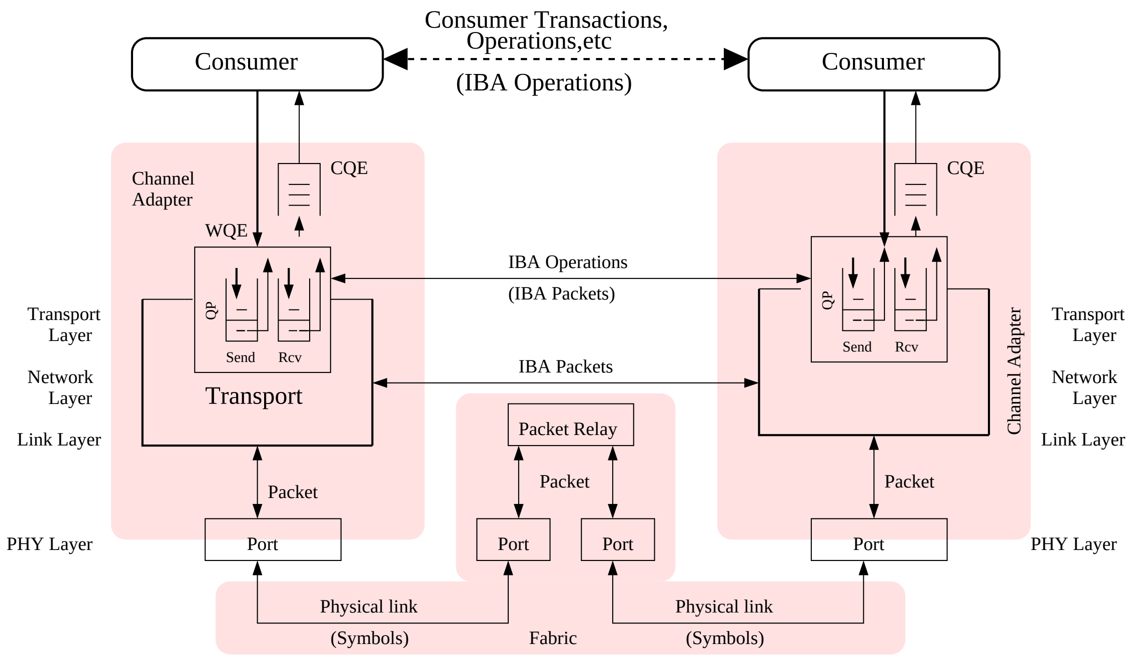

Figure 2.2: InfiniBand Architecture (Courtesy InfiniBand Specifications)

2.2.1 IB Communication

Each Queue Pair is a communication endpoint. A Queue Pair (QP) consists of the send queue and the receive queue. Two QPs on different nodes can be connected to each other to form a logical bi-directional communication channel. An application can have multiple QPs. Communication requests are initiated by posting Work Queue Entries (WQEs) to these queues. Each WQE is associated with one or more pre-registered buffers from which data is either transferred (for a send WQE) or received (receive WQE). The application can either choose the request to be a Signaled (SG) request or an Un-Signaled request (USG). When the HCA completes the processing of a signaled request, it places an entry called as the Completion Queue Entry (CQE) in the Completion Queue (CQ). The consumer application can poll on the CQ associated with the work request to check for completion. There is also the feature of triggering event handlers whenever a completion occurs. For un-signaled requests, no kind of completion event is returned to the user. However, depending on the implementation, the driver cleans up the Work Queue Request from the appropriate Queue Pair on completion.

2.2.2 RDMA Communication Model

IB supports two types of communication semantics: channel semantics (send-receive communication model) and memory semantics (RDMA communication model).

In channel semantics, every send request has a corresponding receive request at the remote end. Thus, there is a one-to-one correspondence between every send and receive operation. Failure to post a receive descriptor on the remote node results in the message being dropped and retransmitted for a user specified amount of time.

In the memory semantics, Remote Direct Memory Access (RDMA) operations are used. These operations are transparent at the remote end since they do not require the remote end to involve in the communication. Therefore, an RDMA operation has to specify both the memory address for the local buffer as well as that for the remote buffer. There are two kinds of RDMA operations: RDMA Write and RDMA Read. In an RDMA write operation, the initiator directly writes data into the remote node's user buffer. Similarly, in an RDMA Read operation, the initiator directly reads data from the remote node's user buffer.

Most entries in the WQE are common for both the Send-Receive model as well as the RDMA model, except an additional remote buffer virtual address which has to be specified for RDMA operations.

2.2.3 Atomic Operations Over IB

In addition to RDMA, the reliable communication classes also optionally support atomic operations directly against the memory at the end node. Atomic operations are posted as descriptors at the sender side as in any other type of communication. However, the operation is completely handled by the NIC and involves very little host intervention and resource consumption.

The atomic operations supported are Fetch-and-Add and Compare-and-Swap, both on 64-bit data. The Fetch and Add operation performs an atomic addition at the remote end. The Compare and Swap is used to compare two 64-bit values and swap the remote value with the data provided if the comparison succeeds.

2.3 Sockets Direct Protocol

The SDP standard focuses specifically on the wire protocol, finite state machine and packet semantics. Operating system issues, etc., can be implementation specific. It is to be noted that SDP supports only SOCK_STREAM or streaming sockets semantics and not SOCK_DGRAM (datagram) or other socket semantics.

SDP's Upper Layer Protocol (ULP) interface is a byte-stream protocol that is layered on top of IB or iWARP's message-oriented transfer model. The mapping of the byte stream protocol to the underlying message-oriented semantics was designed to enable ULP data to be transfered by one of two methods: through intermediate private buffers (using a buffer copy) or directly between ULP buffers (zero copy). A mix of send/receive and RDMA mechanisms are used to transfer ULP data.

The SDP specification also suggests two additional control messages known as *Buffer Availability Notification* messages, viz., *source-avail* and *sink-avail* messages for performing zero-copy data transfer.

Sink-avail Message: If the data sink has already posted a receive buffer and the data source has not sent the data message yet, the data sink does the following steps: (i) registers the receive user-buffer (for large message reads) and (ii) sends a *sink-avail* message containing the receive buffer handle to the source. The data source on a data transmit call, uses this receive buffer handle to directly RDMA write the data into the receive buffer.

Source-avail Message: If the data source has already posted a send buffer and the available SDP window is not large enough to contain the buffer, it does the following two steps: (i) registers the transmit user-buffer (for large message sends) and (ii) sends a *source-avail* message containing the transmit buffer handle to the data sink. The data sink on a data receive call, uses this transmit buffer handle to directly RDMA read the data into the receive buffer.

Based on the details provided in the standard, there have been several implementations of SDP. Specifically, for the buffer copy based implementations of SDP, communication is carried out in intermediate private buffers using flow-control mechanisms such as credit-based flow control and packetized flow control [6], each having its own limitations.

For the zero-copy based implementations of SDP, the *source-avail* and *sink-avail* based mechanism described in the standard is most relevant only for asynchronous sockets due to their capability of exposing multiple source or sink buffers simultaneously to the remote node. Accordingly, most current implementations for synchronous sockets do not implement these and use only the buffer copy based scheme. Recently, Goldenberg et. al., have suggested a zero-copy SDP scheme [25, 24]. In this scheme, they utilize a restricted version of the *source-avail* based zero-copy communication model for synchronous sockets. Due to the semantics of the synchronous sockets, however, the restrictions affect the performance achieved by zero-copy communication significantly. Further, a more advanced zero-copy scheme with asynchronous communication pattern for synchronous sockets has been suggested by Balaji et. al [7]. This scheme improves performance of SDP by allowing multiple outstanding messages.

CHAPTER 3

DESIGNING THE SOCKETS DIRECT PROTOCOL (SDP) OVER IWARP

As mentioned earlier, the SDP standard deals with IB and iWARP uniformly. This means that, the SDP implementation has to be independent of the underlying network communication stack. In reality, however, IB and iWARP are vastly different. IB has been designed mainly for SAN environments, whereas iWARP is aimed to bring the SAN capabilities (e.g., RDMA) to Ethernet-based WAN environments. The way each of these network stacks is implemented could affect the performance of SDP vastly.

Specifically, in the case of iWARP, several implementations exist. These implementations can be broadly categorized into WAN compatible iWARP implementations and WAN-incompatible implementations. WAN-compatible iWARP implementations are completely compatible with the general TCP/IP based WAN environment. WAN-incompatible iWARP implementations, on the other hand, give up such compatibility to some extent in order to achieve a better performance.

Several WAN compatible iWARP implementations have been proposed earlier. Each of these implementations has its own limitations. The performance of an upper layer such as SDP depends on the performance of the underlying iWARP

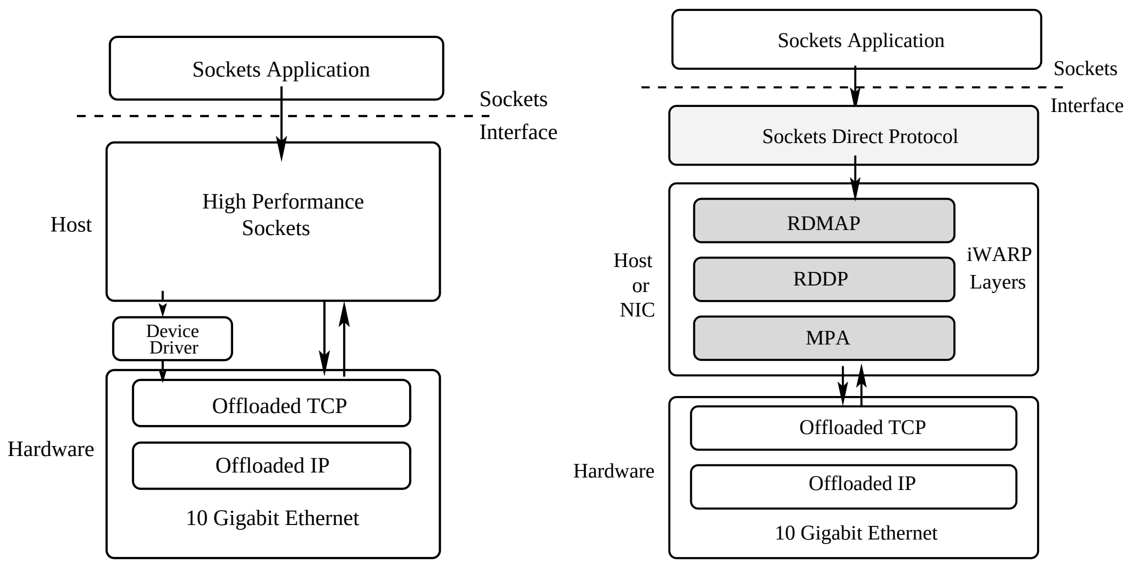

Figure 3.1: (a) 10 Gigabit Ethernet TOE stack (b) iWARP protocol stack

implementation. For this reason, it is important to understand the different designs that are available for iWARP. We will discuss these in Section 3.1.

In this chapter, we first describe the design of a generic SDP implementation over iWARP and show its performance. We highlight the limitations of such an implementation and describe the reasons for its low performance. In Section 3.2.2, we propose a new integrated framework for SDP and iWARP which overcomes the performance limitations of the generic SDP/iWARP design while still maintaining complete WAN compatibility. Section 3.3 shows the performance of various iWARP implementations and that of SDP/iWARP. We provide a summary in Section 3.4.

3.1 Design Choices for iWARP

In this section, we briefly describe the different design choices for iWARP implementation over Ethernet. The existing iWARP implementations can be broadly categorized into (i) WAN-compatible iWARP and (ii) WAN-incompatible iWARP. As discussed in 2.1.2, to achieve WAN compatibility, iWARP's MPA layer is necessary. WAN-compatible iWARP refers to the marker-based design of iWARP which achieves complete WAN compatibility with layer-4 Ethernet devices. WAN-incompatible iWARP, on the other hand, refers to the design of iWARP which does not implement the marker insertion logic which is necessary for WAN compatibility. Such iWARP enabled adapters are compatible only up to layer-3.

For completely WAN compatible iWARP, three design choices exist. They are (i) Software iWARP, (ii) NIC-offloaded iWARP and (iii) Host-assisted iWARP.

1. **Software iWARP Implementation:** This is a completely host-based implementation of iWARP. It is a generic implementation which can be used on any Ethernet adapter while maintaining complete compatibility with hardware iWARP implementations. There are several design aspects associated with improving the performance of this implementation, which have been described in [8].

2. **NIC-offloaded iWARP Implementation:** This is an implementation of iWARP that is completely based on the hardware and firmware present on the network adapter. This approach is similar to that taken by most modern SANs such as InfiniBand, Myrinet [3], etc., for providing RDMA capabilities.

3. **Host-assisted iWARP Implementation:** This is a hybrid implementation which takes the best characteristics of both the software iWARP implementation as well as the NIC-offloaded iWARP implementation. Specifically, host-assisted iWARP performs compute intensive tasks such as CRC-based data integrity (part of MPA), connection demultiplexing and packet processing (part of RDDP) completely on the network adapter using dedicated processing engines (e.g., CRC engines), while retaining tasks such as marker insertion (part of MPA) in the host space.

In both our SDP/iWARP designs, we use host-assisted iWARP. As mentioned earlier, host-assisted iWARP performs marker insertion in software. It inserts markers into the data stream, while copying it into a temporary buffer. Though message copy is expensive for large message sizes, the NIC now can DMA larger chunks of data and send them out on the network, resulting in higher throughput. This is the reason for choosing Host-assisted iWARP for our SDP/iWARP implementations.

3.2 Sockets Direct Protocol over iWARP

As mentioned in Section 2.3, several implementations of SDP/InfiniBand already exist. However, there has been no previous study on SDP over iWARP. The first scheme for SDP/iWARP implementation described in this section is similar to the existing SDP/IB implementations. We then describe the drawback with this implementation of SDP/iWARP and propose an integrated SDP/iWARP framework which is presented in Section 3.2.2.

3.2.1 Generic SDP/iWARP design

SDP is a byte-stream transport protocol that closely mimics TCP's stream semantics. SDP is an industry standard specification for InfiniBand and iWARP; it utilizes their advanced capabilities such as protocol offload, kernel bypass, and zero copy capabilities to achieve high performance without requiring any modifications to the application. Because of this, SDP can achieve lower CPU and memory bandwidth utilization when compared to conventional implementations of sockets over TCP/IP, while preserving the familiar byte-stream oriented semantics upon which most current network applications depend. This SDP implementation is intended to emulate sockets semantics over TCP/IP, and to be layered on top of iWARP mapped over TCP/IP. This layering is shown in Figure 3.2.

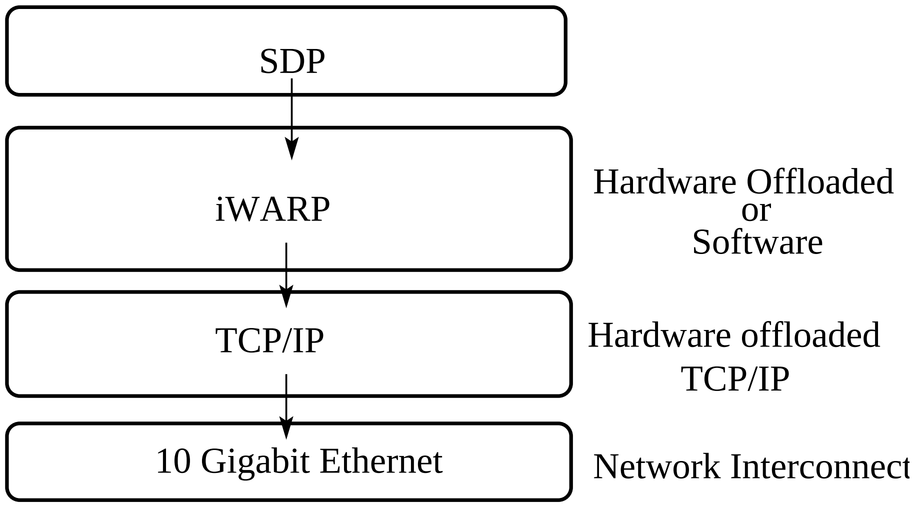

Figure 3.2: Sockets Direct Protocol over iWARP

SDP's Upper Layer Protocol (ULP) interface is a byte-stream protocol that is layered on top of iWARP's message-oriented transfer model. This mapping is designed so as to enable ULP data to be transfered by one of two methods: through intermediate private buffers (using a buffer copy) or directly between ULP buffers (zero copy). In this thesis, we only concentrate on a buffer copy based implementation and defer the zero-copy implementation for future work.

In order to implement SDP over iWARP, several design challenges had to be tackled. These designs are similar to the SDP implementations over InfiniBand and can be found in [10]. We summarize a few of them in this section.

Credit-based flow control: The iWARP specification requires the receiver to pre-advertise a receive buffer before the sender can send the corresponding data. However, this requirement is not a part of the sockets specification. Accordingly, a sockets application might send data out before the receiver advertises the receive buffer. This constraint is known as the *buffer pre-posting constraint*. In order to handle this constraint, we use a credit based mechanism as described in Section 4.1.1

One problem with applying this algorithm directly is that the acknowledgment message also uses up a temporary buffer and hence a credit. Thus, if the number of credits is too few, this might result in a livelock. In order to avoid this livelock, we use the concept of delayed acknowledgments (explained later in this section) where we avoid sending acknowledgments for every free receive buffer, but rather coalesce multiple of these messages and send them together in one message.

Handling fork(): According to the semantics of TCP/IP sockets, if a parent process has an open socket connection, when it forks another child process, both the parent as well as the child process can read or write on the open socket. In order to tackle this problem, we use a token based approach, i.e., every socket has a token. Only the process which holds the token can read or write to the socket. Now, if the parent process has the token and the child process wants to read from or write to the socket, it needs to request the parent process for the token and wait to receive it before performing and socket operation. Note that this issue has a number of corner cases which we had to deal with, e.g., if the parent process has already read data into its temporary buffers and has to give up its token to the child process, it needs to transfer all the data it read to the child process as well.

Delayed Acknowledgments: This is essentially an enhancement to the credit-based flow-control approach. In this approach, in order to improve the throughput achieved, instead of sending an acknowledgment for every data message received, we send an acknowledgment when half the credits are used up. This ensures that there is lesser traffic on the network, lesser work to be done by the protocol stacks and higher performance for the applications.

3.2.2 Design of Integrated SDP/iWARP Framework

As we will see in Section 3.3, the generic SDP implementation suffers from mismatches in the various layers of the stack. SDP is a byte-stream protocol, i.e., it does not deal with message boundaries and tries to coalesce messages, while iWARP is a message-oriented protocol, i.e., it tries to preserve the message boundaries and adds additional processing overhead (e.g., marker insertion) in order to deal with the streaming-based TCP/IP protocol. In other words, the SDP/iWARP stack moves data from the streaming-based SDP stack to the message-oriented iWARP stack and finally back to the streaming-based TCP/IP stack, adding additional overhead at each layer.

The SDP specification forces the implementation of SDP to be decoupled from the iWARP implementation, i.e., the iWARP implementation cannot assume that its upper layer is SDP and SDP cannot assume any details about the implementation of the iWARP layer. While this is understandable from the stand-point of generality, it is not the best for achieving high performance. For a clearer understanding of this overhead, we propose an integrated SDP/iWARP stack in this Section. The main idea of this design is that the iWARP stack provides an extended interface for the SDP ULP and the SDP ULP can assume that the underlying protocol is host-assisted iWARP.

Specifically, the SDP implementation does not perform any buffering of the data on the sender side, but rather passes on the data directly to the iWARP stack. The iWARP stack (host-assisted) performs the actual buffering of the data while simultaneously utilizing this buffering operation to insert markers at appropriate locations. Similarly, on the receiver side, the iWARP stack does not remove the markers from the received data, but rather hands over the entire data segment (together with strips of markers in between the data stream) to the SDP layer. The SDP layer removes the markers before handing over the data to the application. In summary, the integration of the SDP/iWARP stack allows to reduce the buffering requirements and memory transactions of the stack and thus improve the performance.

Performance results comparing the integrated stack with the generic SDP implementation as well as with SDP/iWARP without markers are shown in Section 3.3.3.

3.3 Performance Results

In this section we describe our evaluation framework and compare the performance of three different iWARP designs. Further, we also evaluate the Integrated SDP and iWARP implementation and compare it with the generic SDP design over iWARP.

3.3.1 Experimental testbed

For the experimental testbed, we used a cluster of four nodes built around SuperMicro SUPER X5DL8-GG motherboards with ServerWorks GC LE chipsets, which include 133-MHz PCI-X interfaces. Each node has two Intel Xeon 3.0 GHz processors with a 512-KB cache and a 533 MHz front-side bus and 2 GB of 266-MHz DDR SDRAM.

The nodes are connected with Chelsio T110 10-Gigabit Ethernet TCP Offload Engines through a 12-port Fujitsu XG800 cut-through switch. The driver version on the network adapters is 1.2.0.

3.3.2 iWARP Evaluation

In this section, we evaluate the three different mechanisms proposed to implement iWARP using standard ping-pong latency and unidirectional bandwidth micro-benchmarks.

Figure 3.3 shows a comparison of the ping-pong latency of the three designs. For small messages, all three schemes perform similarly at about 16μs. However, as the message size increases software iWARP performs the worst amongst all the designs. This is expected as this design does not take advantage of any advanced hardware

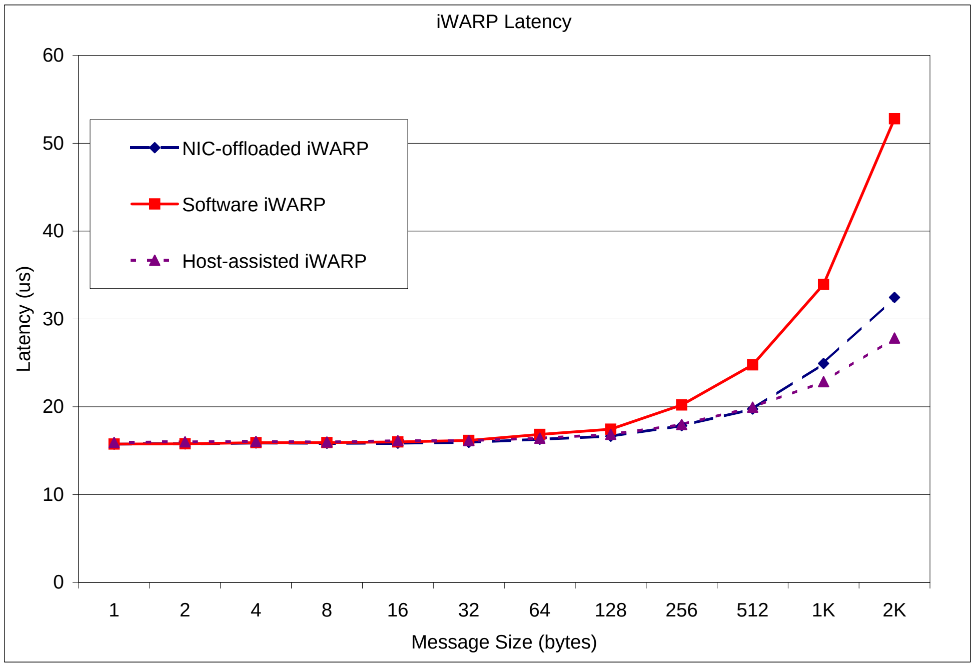

Figure 3.3: iWARP Latency

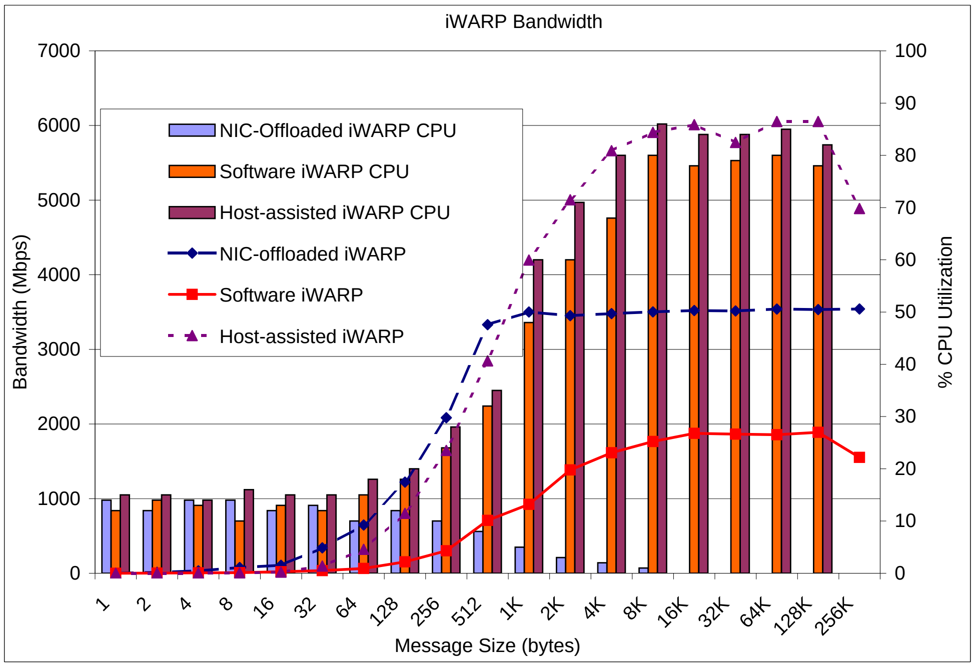

Figure 3.4: iWARP Bandwidth

present on the network adapter and performs all operations in software. Comparing NIC-offloaded iWARP and host-assisted iWARP, we notice that the performance of NIC-offloaded iWARP deteriorates with increasing message size and it is outperformed by the host-assisted iWARP implementation. This trend is attributed to the overhead of multiple DMAs on NIC-offloaded iWARP, e.g., for a 2KB message, the host-assisted iWARP performs just one DMA operations, while with a standard marker separation length of 512 bytes NIC-offloaded iWARP needs to perform 5-6 DMA operations.

Figure 3.4 shows a comparison of the uni-directional bandwidth of the three designs. The basic trend for this result is also quite similar to the ping-pong latency performance. The software iWARP implementation again performs the worst achieving a throughput of only about 2Gbps. This is attributed mainly to the overhead associated with copying data as well as the CRC data-integrity check, which increases linearly with the message size. Comparing NIC-offloaded iWARP and host-assisted iWARP, for very small messages NIC-offloaded iWARP performs slightly better than host-assisted iWARP – we are currently analyzing this behavior to understand its reasoning. For the peak throughput, however, host-assisted iWARP outperforms the NIC-offloaded iWARP with throughputs of about 6Gbps and 3.5Gbps, respectively. The reason for the performance limitation of NIC-offloaded iWARP is again the number of DMA operations, i.e., according to the design of NIC-offloaded iWARP, it can perform DMA operations of only about 512 bytes in each operation. Though, pipelining the DMA operations can improve the performance a little, it is eventually limited by the DMA overhead. Host-assisted iWARP, on the other hand, can DMA full length 2KB data chunks in each operation and thus can achieve a higher performance.

Figure 3.4 also shows the CPU utilized by the different iWARP implementations. As shown in the figure, NIC-offloaded iWARP uses the least amount of host-CPU using less than 15% for all message sizes and practically 0% for large messages. This is expected since this is a complete hardware offloaded implementation; further, being a zero-copy implementation, for large messages, the message transfer initiation time (done by the host) is negligible compared to the actual transfer time. The high CPU utilization of software iWARP is also expected since it performs all tasks, including highly compute intensive tasks such as the CRC calculation, in software. The surprising thing, however, is that the CPU utilization of the host-assisted iWARP is even higher than the software iWARP implementation. This is attributed to the higher performance of host-assisted iWARP. Note that the host-assisted iWARP implementation performs a copy of the data into a temporary buffer before transmitting it in order to insert the markers at appropriate locations. Now, since the performance of host-assisted iWARP is higher than that of software iWARP, the data is sent out of the temporary buffer quickly, thus requiring the CPU to spend a larger fraction of the time performing the memory copy operation; this results in a higher CPU utilization.

3.3.3 SDP evaluation

In this section, we evaluate the SDP implemented on top two different versions of iWARP, viz., iWARP with markers and iWARP without markers. That is, we compare the performance of SDP over WAN compatible iWARP with that of SDP over WAN-incompatible iWARP.

As shown in Figure 3.5, the implementation of SDP over WAN-incompatible iWARP achieves a slightly better latency of 15μs as compared to the 16μs achieved by SDP over WAN-compatible iWARP. Also, Figure 3.6 shows that the bandwidth achieved by the SDP/iWARP with markers is close to 5.2Gbps as compared to the 6.4Gbps achieved by SDP/iWARP (no markers).

As described in Section 3.2.2, the primary reason for this lower performance is the additional overhead added by the MPA layer for marker insertion. SDP/iWARP (no markers), on the other hand, does not implement any kind of marker insertion logic and hence avoids significant overhead. Effectively, by giving up WAN-compatibility, SDP/iWARP is able to achieve better performance than SDP/iWARP (with markers). In Figures 3.5 and 3.6, we present the performance of the integrated SDP/iWARP stack that is described in Section 3.2.2 as compared to the SDP/iWARP (no markers).

Figure 3.7 compares the ping-pong latency of the integrated SDP/iWARP stack with that of SDP/iWARP with markers and SDP/iWARP without markers. As the figure indicates, both integrated SDP/iWARP and SDP/iWARP without markers achieve similar performance of about 15μsec. That is the integrated SDP/iWARP framework achieves better performance than the generic SDP/iWARP implementation without sacrificing WAN compatibility. The reason is as follows: In integrated SDP/iWARP, buffering of data by the SDP layer and insertion of markers by iWARP layer are combined into a single operation. Thus, we reduce one copy on the sender

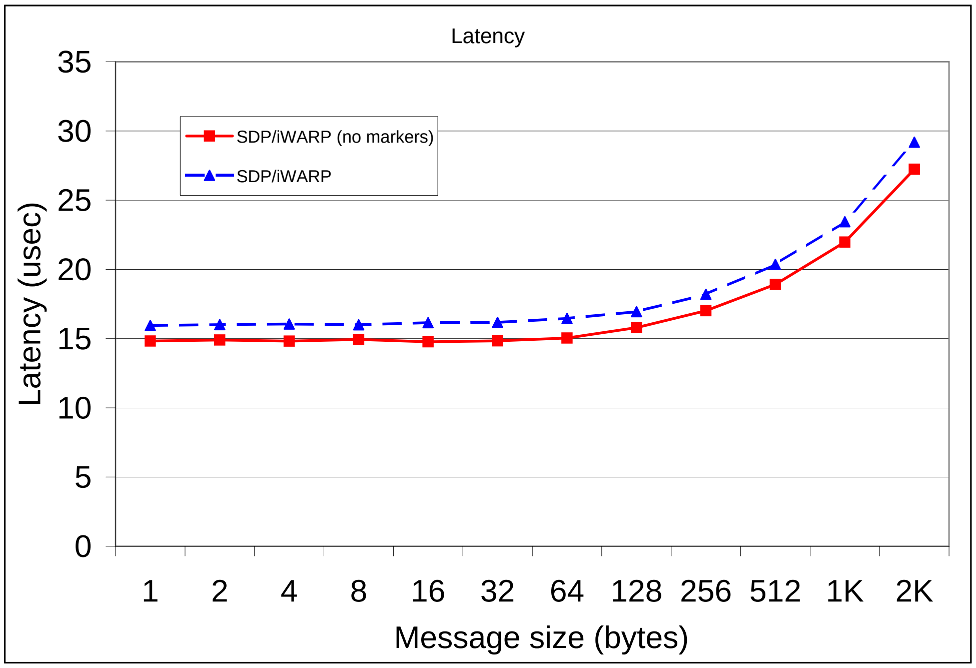

Figure 3.5: SDP/iWARP (with markers) and SDP/iWARP (no markers): Latency

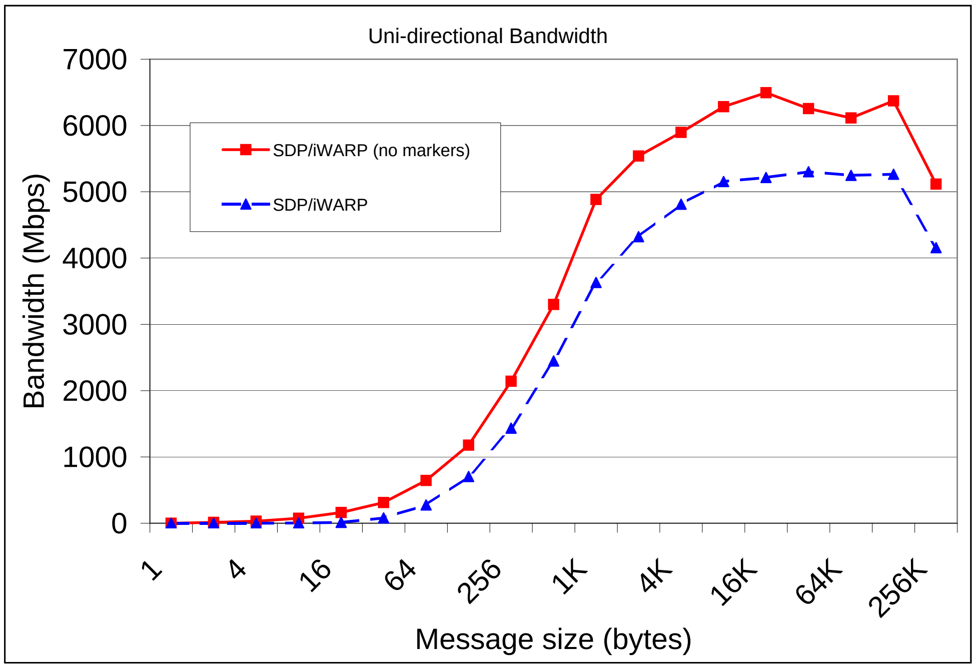

Figure 3.6: SDP/iWARP (with markers) and SDP/iWARP (no markers): Bandwidth

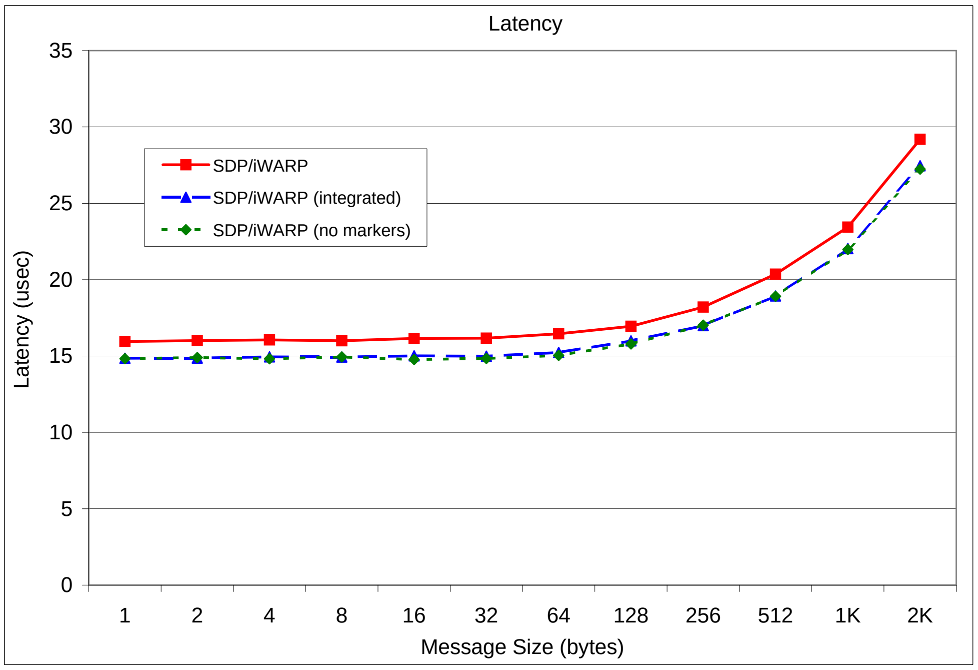

Figure 3.7: SDP Micro-benchmarks: Latency

side. On receiver side, iWARP does not remove the markers. The SDP layers removes the markers while it buffers the data. Thus, each message transfer takes two lesser memory copies there by resulting in better performance.

In figure 3.8, we present the performance of the three SDP/iWARP stacks. It can be seen that the uni-directional throughput of both integrated SDP/iWARP and SDP/iWARP (no markers) is very similar, confirming that integrated SDP/iWARP achieves better performance than the generic SDP/iWARP and almost same performance as SDP/iWARP (no markers) without the loss of WAN compatibility.

It is to be noted that through the integration, we can gain some performance, but lose out on generality. However, more importantly, we notice that for ULPs like SDP which do not expose the advanced feature set of iWARP (e.g., RDMA), an

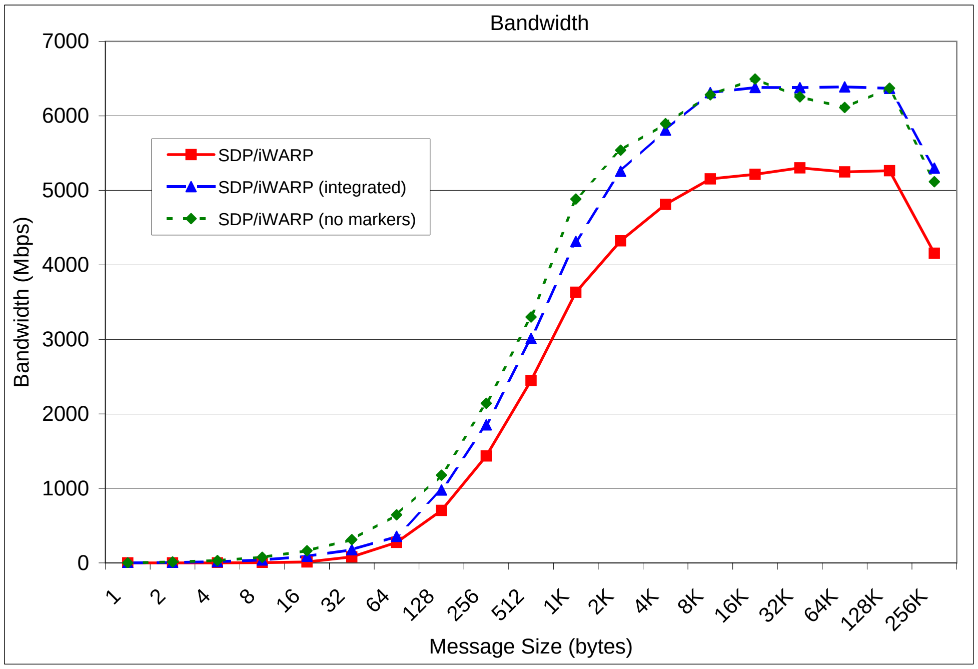

Figure 3.8: SDP Micro-benchmarks: Bandwidth

implementation over iWARP does not gain any significantly hike in performance. In fact, due to the compatibility overheads added by the iWARP stack, utilizing the TOE directly without going through the iWARP stack might be a better solution. Having said that, we also point out that for ULPs which do utilize the advanced features of iWARP (e.g., one-sided communication in MPI), an implementation on top of iWARP instead of directly on TOEs might be better.

3.4 Summary

The Internet Wide Area RDMA Protocol (iWARP) is an industry standard specification for achieving the capabilities of traditional system-area networks for Ethernet, while maintaining wide-area network (WAN) compatibility. In this chapter, we first designed a generic SDP/iWARP implementation and showed that to maintain WAN compatibility, the SDP/iWARP stack has to sacrifice some performance. Also, due to mismatch between SDP stack and the iWARP stack, several data copy operations are performed, which affect the performance. Next, we designed an integrated framework for SDP and iWARP which avoids the repetition of processing in the SDP and iWARP layers and achieves better performance while maintaining complete WAN compatibility. Our experimental results have shown that Integrated SDP/iWARP performs close to 20% better than the generic SDP/iWARP implementation.

CHAPTER 4

DESIGNING NIC-ASSISTED FLOW CONTROL FOR THE SOCKETS DIRECT PROTOCOL (SDP) OVER INFINIBAND

Most high-speed networks have strict requirements on their upper layers that the receiver has to post a receive descriptor informing the network adapter about where to place the incoming message before the message is actually sent by the sender. Not doing so might result in limited retransmissions of the message (increasing the network load) and/or the connection being dropped or terminated. This is referred to as the *buffer pre-posting constraint*. Thus, most programming models and upper layers use different flow control mechanism to handle this requirement.

Flow control mechanism can be divided into two classes: (i) strong flow control mechanisms and (ii) weak flow control mechanisms. In the strong flow control mechanism, the sender *NEVER* sends data before the receiver is ready to receive it. Most programming models and other upper layers use this kind of flow-control. On the other hand, weak flow control mechanisms relax the *buffer pre-posting constraint*. In weak flow control, the sender assumes that the receiver has enough buffers posted and keeps sending messages. The receiver, on the other hand, asynchronously tries to keep up with the sender. The SRQ-based design of MPI [40, 41, 32], which is an example of weak flow control, implements a scheme where every time the receiver runs out of credits, an interrupt is generated. During the interrupt handling, more receiver buffers are posted. In this thesis, we will focus only on strong flow control schemes.

There are several strong flow control mechanisms that have been proposed earlier. Amongst these, credit-based flow control is the most commonly used flow control mechanism (used in MPI, file systems, high performance sockets, etc.). However, as we will see in Section 4.1.1, while this scheme is simple, it could result in severe under-utilization of resources and performance degradation. In order to meet the limitations of credit-based flow control, Balaji et. al., recently proposed a new mechanism known as packetized flow control. Packetized flow control utilizes the RDMA capabilities of IB to improve resource utilization. It also improves performance through coalescing of data and there by sending larger messages on the network. This scheme, however, could lead to indefinite lack of communication progress.

In this chapter, we propose a new flow-control mechanism known as NIC-assisted flow control. This mechanism utilizes the hardware flow-control capabilities of IB to maintain the performance of packetized flow control and allow for better communication progress. We propose several novel designs for achieving this including: (a) Virtual window mechanism and (b) Asynchronous Interrupt-based mechanism. We also show preliminary results for our design, where our scheme achieves a similar throughput as packetized flow control, while achieving close to 20% better communication progress performance.

The rest of the chapter is arranged as follows. Section 4.1 gives an overview of the existing flow-control techniques. We then present the detailed design of NIC-assisted flow-control mechanism in Section 4.3. Experimental results showing these schemes is given in Section 4.4. A brief summary is provided in Section 4.5.

4.1 Overview of Existing Flow Control Mechanisms

To handle the *buffer pre-posting constraint*, most programming models and upper layers use the credit-based flow-control approach. We give a detailed description of this scheme along with its limitations in sections 4.1.1 and 4.1.2.

Recently, a new flow-control mechanism for SDP known as the packetized flow control has been proposed, which utilizes RDMA-based one-sided communication operations to perform complete sender-side buffer management for both the sender as well as the receiver buffers. This allows the user to remove the "communication gap" that is formed between the sender and the receiver for managing buffers and helps us in improving the buffer usage as well as the performance achievable by SDP. Packetized flow control mechanism and its limitations are described in sections 4.1.3 and 4.1.4.

4.1.1 Credit-based Flow-control in SDP

As mentioned earlier, most programming models and upper layers use the credit-based flow control mechanism for communication. Here we provide a brief overview of the same.

In this approach, the sender is given a certain number of credits (tokens). It loses a credit for every message sent and gains a credit for every acknowledgment received. If the sender is given N credits, the SDP layer has to make sure that there

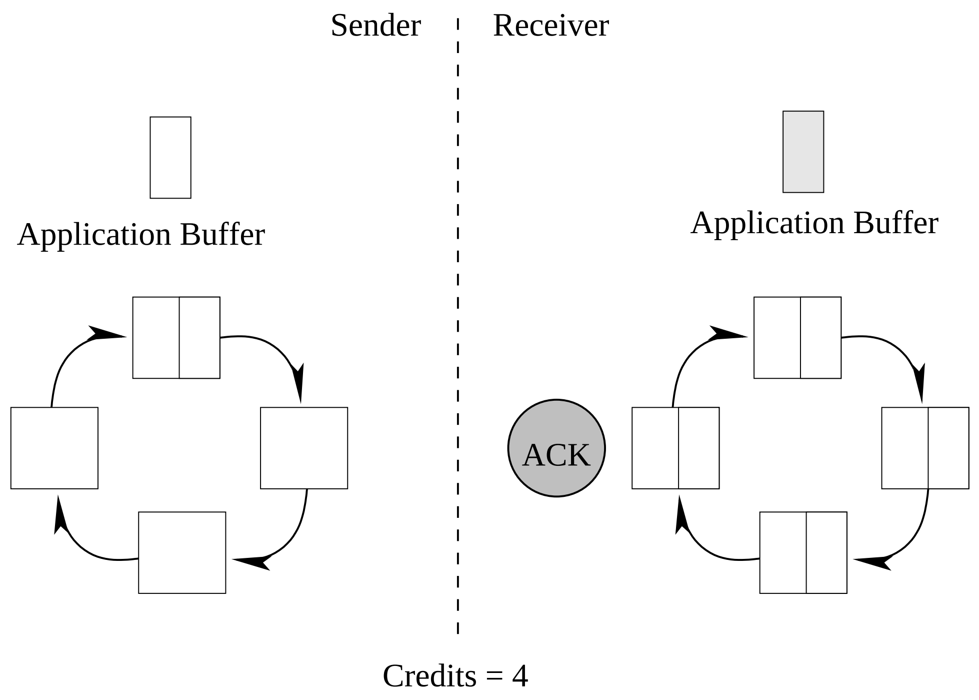

Figure 4.1: The Credit Based Approach

are enough descriptors and buffers pre-posted for N unexpected message arrivals on the receiver side. In this way, the substrate can tolerate up to N outstanding `send()` calls before the corresponding `recv()` for the first `send()` is called (Figure 4.1).

In the credit-based flow control approach, the receiver is *blind* to the sizes of the incoming messages. Accordingly, statically sized buffers (of size S) are allocated in circular fashion, both on the sender and the receiver side. There are as many intermediate buffers, of size S, as the number of credits. So, if there are credits available, the sender can directly copy the outstanding message into the intermediate buffer and send it out to the next available receive buffer. If the message is S bytes or smaller in size, it is copied to the intermediate buffer on the sender side and sent to the receiver side intermediate buffer. If the message is larger than S bytes in size, it is broken up into S byte chunks and copied into as many intermediate buffers as available. Data from each of these buffers is sent out as soon as a credit is available.

4.1.2 Limitations with Credit-based Flow Control

The main disadvantage of the credit-based flow control scheme is based on the way it handles the communication of small messages, i.e., when the sender is transmitting small messages, each message uses up an entire buffer on the receiver side, thus wasting the buffer space available. For example if each message is only 1 byte and each buffer is 8 KB, effectively 99.98% of the buffer space is un-utilized. This wastage of buffers also reflects on the number of messages that are sent out, i.e., excessive under-utilization of buffer space might result in the SDP layer to *believe* that it has used up its resources in spite of having free resources.

Another disadvantage of the credit-based flow control mechanism is its network utilization. Since this approach directly sends out data as soon as the sender has requested for transmission, it might result in very small messages being posted to the network. This, of course, results in the under-utilization of the network and hence in degradation in performance.

4.1.3 Packetized Flow control in SDP

As described in the previous section (Section 4.1.1), the credit-based flow control mechanism has its own disadvantages. Packetized flow-control has been designed to solve these problems. In this section, we provide the design and implementation details of the same.

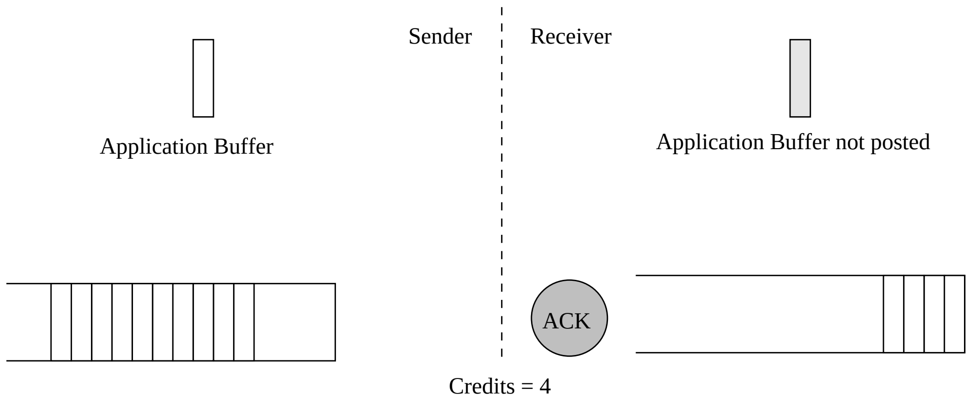

Figure 4.2: Packetized Flow Control Approach

Packetized flow-control (Figure 4.2) utilizes advanced network features such as RDMA to tackle the limitations of credit based flow control. In this scheme, the entire intermediate buffer is one continuous buffer instead of several buffers connected in a circular list. Or in other words, the intermediate buffer is packetized into buffers of 1 byte size. The entire buffer management, for both the sender as well as the receiver, is carried out on the sender side alone using RDMA operations. Since the sender knows exactly what size messages it is sending, it can manage the receiver buffer in a better manner. When the new message has to be sent out, the sender knows the address of the next free location on the receiver side and can place the new message in the appropriate position using an RDMA write operation. Thus the wastage of buffers is minimal in this approach and close to the ideal 100% in most cases.

Further, if the application posts a new message to be sent after all the credits have been used up, the message can be copied to the intermediate buffer (where there is more space available due to better buffer management) and sent at a later time. The application `send()` call returns allowing the application to carry on with its computation. This allows the SDP layer to coalesce multiple small messages into one larger message, thus improving network utilization and hence performance.

In summary, the advantages of Packetized flow control are two fold. First, it avoids buffer wastage for small and medium sized messages. Sender side buffer management assures that there is no buffer wastage. Second, it increases throughput for small and medium sized messages by coalescing small messages when the sender is out of remote credits.

4.1.4 Limitations with Packetized Flow Control

While the packetized flow-control mechanism discussed in the previous section (Section 4.1.3), can achieve a better resource utilization and a high performance, it has certain disadvantages such as indefinite lack of communication progress in some cases. We describe this limitation in this section.

The design of packetized flow control is completely sender based, i.e., sender performs buffer management for both the sender and the receiver. The receiver is only involved in receiving the data into the application buffer. Consider an example where the sender has initiated a large number of message sends. The sockets buffer is typically 64KB in size. Suppose, the sender initiates a data transfer of 128KB. Of this, 64KB is directly posted to the network to be buffered on the receiver side. After this, assuming that the receiver is not actively receiving data, the sender will run out of credits. Thus, the remaining 64KB is copied to the intermediate buffer and the control is returned back to the application. Now, suppose the application on the sender side goes into a computation loop, and the receiver sends an acknowledgment during this time. In this situation, the sender has both remote credits as well as data to send (in its intermediate buffer). However, until the application comes out of its computation loop and calls a communication function, no progress can be made.

Note that credit-based flow control does not face this limitation since for every data transmission call, if the sender does not have any credits, it blocks till credits are received and posts the data to the network before returning the control to the application.

4.2 Overview of InfiniBand Message Level Flow control

We provide a brief overview of the flow control mechanism provided by InfiniBand hardware in this section.

IB provides a message level end-to-end flow control capability for reliable connections that can be used by a receiver to optimize the use of its receive resources. Essentially, a sender cannot send a request message unless it had appropriate credits to do so. Encoded credits are transported to from the receiver to the sender in an acknowledge message.

Each credit represents the receive resources needed to receive one inbound send message. Specifically, each credit represents one WQE posted to the receive queue. The presence of a receive credit does not, however, necessarily mean that enough physical memory has been allocated. For example, it is still possible, even if sufficient credits are available, to encounter a condition where there is insufficient memory available to receive the entire inbound message.

Two mechanisms are defined for transporting credits from the receiver's receive queue to the sender's send queue. The credits can be piggybacked onto an existing acknowledge message, or a special unsolicited acknowledge message can be generated by the receiver. Piggybacked credits are those credits that are carried in an already scheduled acknowledge packets.

1. **Piggybacked Credits:** Piggybacking of end-to-end credits refers to transferring credits to the sender in a normal acknowledge packet. Credits can be piggybacked onto any acknowledge packets with a valid Message Sequence Number (MSN).

2. **Unsolicited Acknowledge packet:** An unsolicited acknowledge message appears to the sender like a duplicate of the most recent positive acknowledge message. Unsolicited acknowledge may be sent by the receiver at any time. The sender's send queue simply recovers the credit field from the most recently received acknowledge packets. Since an unsolicited acknowledge packet appears to the sender as a duplicate response, it has no effect on the sender other than transfer of the credits.

4.3 Design of NIC-assisted Flow Control

Previously proposed flow-control mechanisms, as described in Section 4.1, are entirely implemented in software. While this gives them the additional flexibility of making intelligent choices (e.g., coalescing of data, segmentation), they suffer from various limitations including performance and resource usage (for credit-based flow control) and lack of communication progress (for packetized flow control). IB's hardware flow control, on the other hand, performs the entire flow control in hardware. While this has certain advantages, such as asynchronous (hardware controlled) communication progress, it lacks the intelligent schemes such as coalescing and segmentation of data, that are used by previous mechanisms.

NIC-assisted flow control is a hybrid software-hardware flow control mechanism that performs some tasks in software while taking advantage of the flow control mechanism provided by InfiniBand (described in Section 4.2). There are two main schemes in the NIC-assisted flow control mechanism. The first one is a virtual window based scheme which is used mainly to avoid data corruption and thus maintain correctness. The second scheme is an asynchronous interrupt based scheme. This scheme is an enhancement to the virtual window based scheme to improve the performance by coalescing data. We describe the two schemes in detail.

4.3.1 Virtual Window-based Scheme

Virtual window-based scheme is a mechanism to ensure that the sender never overruns the receiver during flow control. As mentioned earlier, IB's hardware flow control is not byte-level flow control. So, when a sender sees a receive request, it could send a message that is larger than the posted receive buffer, resulting in erroneous behavior. To avoid this, the sender and receiver should agree on what size messages to send and when to send an acknowledgment. For this, we define a *virtual window (W)*.

In this scheme, the sender keeps track of the size of the messages it is trying to send out and the amount of free buffer space available on the receiver side. It ensures that message segments posted to the network are always smaller than or equal to W by performing appropriate segmentation. The receiver, on the other hand does not advertise a free buffer every time it copies a message into application buffer. Instead, it only does so if it has at least W amount of free socket buffer space. If remote credits are not available, the sender copies the messages into the sockets buffer and hands them over to the NIC. When the NIC sees that the receiver has advertised a buffer, it sends the next posted message on the network. Now, since the sender always guarantees that each message it at most W bytes and the receiver always advertises a buffer when its at least W bytes, there is no buffer overflow. Further, since the NIC is handling the communication, progress can be carried on even if the sender is busy computing.

The point to be noted here, however, is that when the sender has no remote credits available, it does not coalesce messages. It copies the message to the temporary buffer and directly posts it to the NIC. Thus, though we are achieving good communication progress, we are sending out a large number of smaller messages on the network which could eventually result in a performance degradation as compared to packetized flow control.

4.3.2 Asynchronous Interrupt-based Scheme

Asynchronous interrupt-based scheme (Figure 4.3), is proposed as an enhancement to the virtual window scheme described in Section 4.3.1.

In this scheme, the sockets buffer is divided into two portions, one portion is handled by the software and another portion is handled by the NIC (this distinction is followed only when the sender side has run out of remote credits). All messages in the NIC handled portion are handed over to the NIC as soon as they are copied.

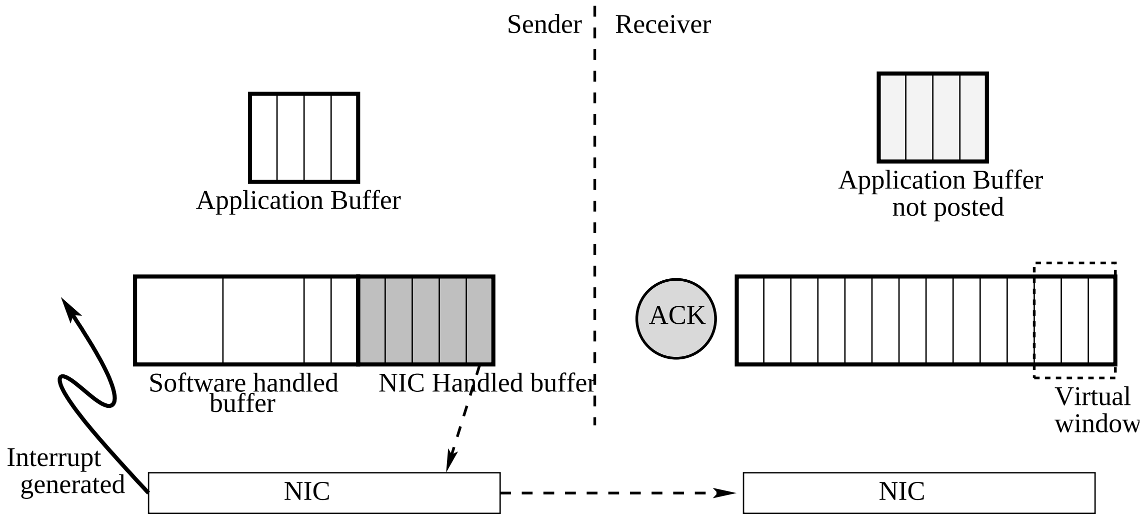

Figure 4.3: NIC-assisted Flow Control Approach

All messages copied into the software handled portion are treated in the same way as packetized flow control, i.e., they are coalesced without handing over to the NIC. Specifically, when the sender is out of remote credits and it has more data to be sent out, it copies the data first to the NIC handled portion of the sockets buffer and posts these messages to the NIC. Messages that arrive after this portion is full are copied to the software handled portion of the buffer.

Considering the example stated earlier, let us suppose that the sender is doing some computation now. As described in the virtual window scheme, the receiver advertises a buffer when there is at least W sized buffer free. The sender sees this sends the next message that was posted to it. This is from the NIC handled portion of the buffer. Simultaneously, it generates an interrupt. The interrupt is handled as follows – the SDP library now looks into the software handled portion of the sockets buffer. It coalesces the messages in this portion to segments of size W and posts these messages to the NIC. After all the messages in the NIC handled portion are sent out, then the sender side NIC can send out these coalesced messages out onto the network instead of just small messages.

The interrupt handling mechanism in this scheme is quite expensive. However, while the software is handling this interrupt, the NIC still has some more messages to be sent out from the portion of the buffer it is handling. The transfer of these messages overlaps with the interrupt handling routine hiding its overhead and allowing for an improved performance. As we will see in the next section, the performance of NIC-assisted flow control is close to that of packetized flow-control, showing that the interrupt handling overhead is overlapped with data transmission and does not affect performance adversely.

4.4 Experimental Results

In this section, we describe our evaluation framework and compare the performance of our NIC-assisted flow control scheme with that of the existing credit-based and packetized flow control mechanisms.

The experimental test-bed consists of four nodes 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 (capable of a link-rate of 20 Gbps) and are connected to a Mellanox MTS-2400, 24-port fully non-blocking DDR switch.

Figure 4.4 shows the ping-pong latency of SDP with the three flow-control mechanisms. As it can be seen, all the three schemes perform almost the same. The reason for this can be explained by the way these schemes differ. These schemes perform

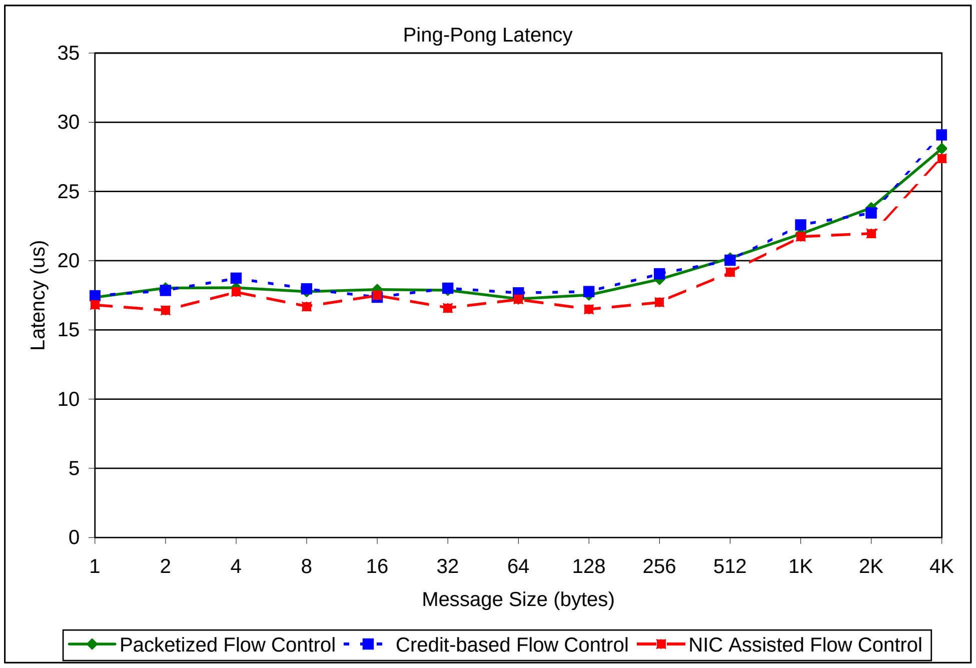

Figure 4.4: SDP/IB with NIC-assisted Flow Control Micro-benchmarks: Latency

differently only when there are no more remote credits available. In the ping-pong latency test, the sender sends one message and waits till it gets a message from the remote side. Thus, the sender and receiver never run out of remote credits during the test and so all the three schemes behave similarly.

In Figure 4.5, we present the performance results for the uni-directional throughput test. As expected, packetized flow control performs the best and credit based flow control performs the worst. The important point to be noted is that NIC-based flow control scheme performs close to that of packetized flow control, i.e., the overhead of interrupt handling in NIC-assisted flow control is overlapped with data transfer time and is almost negligible.

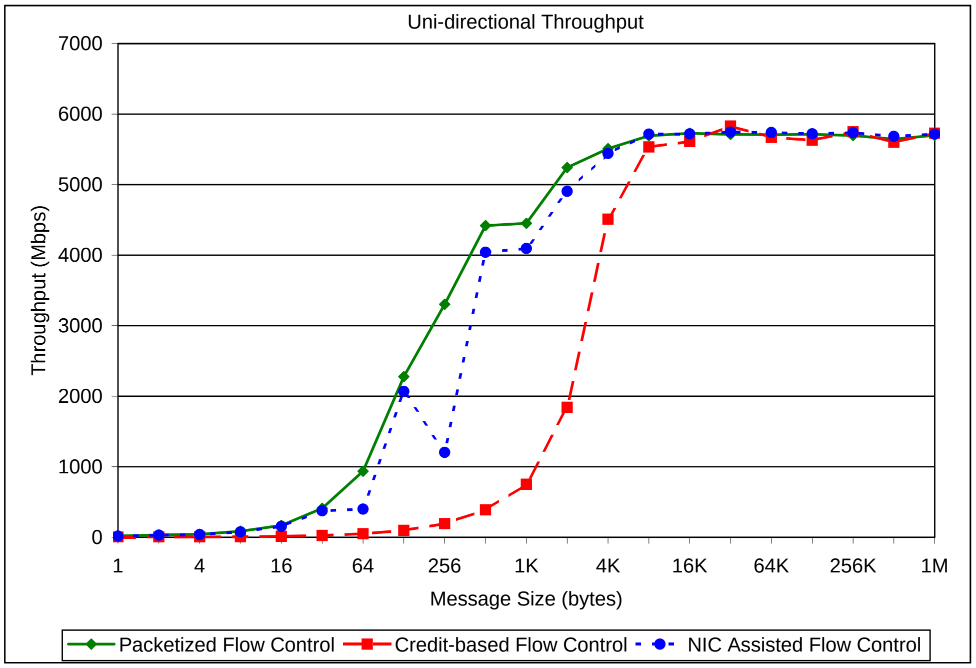

Figure 4.5: SDP/IB with NIC-assisted Flow Control Micro-benchmarks: Bandwidth

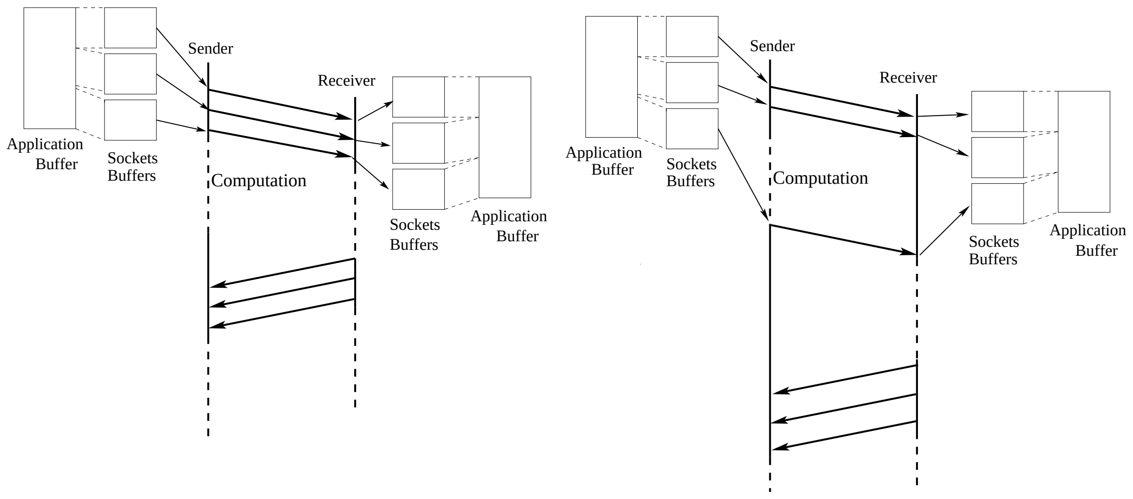

Figure 4.6: (a) Good communication progress; computation loops are highly parallelized and (b) Bad communication progress; computation loops are not parallelized

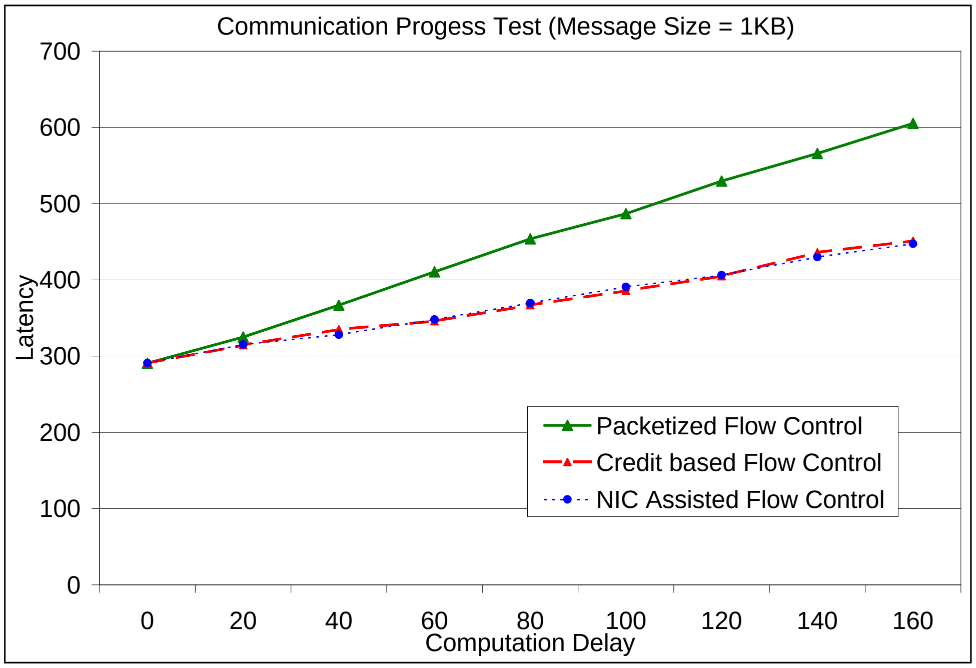

Figure 4.7: SDP/IB with NIC-assisted Flow Control Micro-benchmarks: Computation Communication Overlap Test

The next test is a communication progress test which is similar to a bursty ping-pong latency test to compare the differences between the three flow control mechanisms. The communication progress test is designed as follows. The sender sends a burst of 100 1KB messages to the receiver and then receives 100 messages from the receiver. The receiver first receives 100 messages and then sends 100 messages. Between the send and receive communication operations, however, both the sender and receiver perform computation for a fixed amount of time.

In this test, communication progress is said to be good when the data is sent out as soon as it is copied into the intermediate socket buffers; in this case the computation loops on the sender and receiver side are highly parallelized (Figure 4.6(a)).

Bad communication progress occurs when the data is not sent out as soon as it is copied into the intermediate socket buffers; in this case the sender can copy data and enter a computation cycle. Though the receiver is ready to receive data, till the sender calls another communication call, the buffered data cannot be sent out. In this case, the computation cycles on the sender and the receiver side are completely sequentialized (shown in Figure 4.6(b)).

Figure 4.7 shows the results from the communication progress test. The amount of computation between each burst of communication is varied on x-axis and y-axis shows the latency for each value of computation. As it can be seen, packetized flow control performs the worst with credit based and NIC-assisted performing better and almost equally.

The reason for such results is explained as follows. In credit based flow control, the sender returns from a send call only when it has actually sent the messages on to the network. Communication progress in this case is similar to Figure 4.6. Packetized flow control sends messages onto the network as long as it has remote credits. For sockets buffer of size 64K, the number of 1K messages it can send out is 64. The remaining 36 messages are copied to the intermediate sockets buffer and the sender goes into computation. The receiver, however has received only 64 messages and is waiting for the remaining 36 messages. When the sender completes its computation and is in its next communication cycle, it notices that there are unsent messages and sends these out to the network. After receiving these messages, the receiver now goes into computation while the sender is waiting to receiver messages. This is shown in Figure 4.6(b). In NIC-assisted flow control scheme, the first 64 messages are sent while the remaining 36 messages are buffered and handed over to the NIC to handle. The NIC asynchronously sends out data even when the sender is in a computation cycle, thus allowing for better communication progress.

4.5 Summary

In this chapter, we have discussed the limitations with the existing flow control mechanisms in the current implementations of SDP over InfiniBand. We have also proposed a new flow control mechanism known as NIC-assisted flow control to take advantage of the hardware flow-control mechanism provided by IB. Our experimental results show that NIC-assisted flow control performs close to packetized flow control while achieving better communication progress (up to 25% better in some cases).

CHAPTER 5

CONCLUDING REMARKS AND FUTURE WORK

Traditional communication protocols such as TCP/IP have been implemented in the host kernel space and have not been able to scale with the increasing network speeds of the current high-speed networks. In order to allow existing TCP/IP sockets based applications take advantage of these high-speed networks, researchers have proposed different solutions such as high-performance sockets implementations. The main idea of these high-performance sockets is to provide a pseudo sockets-like interface for end-applications to use while internally utilizing the advanced capabilities of the high-speed networks. The Sockets Direct Protocol (SDP) is an industry standard for such high-performance sockets implementations over InfiniBand and iWARP.

5.1 Summary of Research Contributions

This thesis deals with two main components in SDP: (a) designing SDP over iWARP, studying the various implications associated with such a design and proposing an integrated SDP over iWARP framework to achieve high performance and (ii) designing various enhancements for SDP over InfiniBand to better utilize the hardware support provided by the network adapter.

For the first component, we have first analyzed the capabilities of the traditional design of SDP when applied to iWARP and shown its performance limitations. We have then explored the cause for such performance limitations and have designed an integrated framework with SDP and iWARP which allows a superior performance when compared to the traditional SDP implementation. The performance results, that we showed, demonstrate that the integrated SDP/iWARP framework performs up to 20% better than the basic SDP/iWARP design in some cases.

For the second component, we have studied the limitations of existing flow control mechanisms available for SDP/IB (credit-based and packetized flow control) and proposed a more advanced flow-control mechanism, known as NIC-assisted flow control. This new mechanism retains the benefits of existing mechanisms while efficiently handling their limitations, i.e., it achieves the high-performance of the packetized flow control while addressing its non-optimal communication progress as compared to the credit-based flow control mechanism. Specifically, we take advantage of the hardware flow control mechanism provided by InfiniBand to achieve this. Our performance results show that while achieving nearly similar performance as the packetized flow control scheme, our design shows up to 25% improvement in communication progress.

5.2 Future Research Directions

Apart from the work done in this thesis, there are several aspects that require further research to complete the understanding and analysis of the proposed research area. In this section, we will discuss some of these aspects.

Zero-copy communication for SDP/iWARP: The current implementation of SDP/iWARP only performs buffer-copy based communication, i.e, a copy of the data needs to be made before any communication can be performed. This can result severe negative impact including increased memory traffic, cache pollution and high CPU usage amongst others. One of the major benefits of iWARP compared to TOEs is RDMA, which can result in zero-copy communication for SDP/iWARP, potentially curbing the above mentioned negative aspects. Thus, a zero-copy implementation of SDP/iWARP is amongst the primary extensions to the current work.

Studying other Programming Models over iWARP: The current study only focused on one programming mode, i.e., sockets. However, there are several other widely used programming models such as the Message Passing Interface (MPI), whose interaction with the iWARP stack has not been studied earlier.

Connection Caching in SDP: While SDP has a good data transfer performance, its performance for non data-touching operations such as connection establishment is not the best. For several applications, connection establishment falls in the critical path. Specifically, for every request a connection is established between the client and the server and is torn down at the end of the request. For such applications, caching the connection can improve the performance significantly.

BIBLIOGRAPHY

[1] Mellanox Technologies. `http://www.mellanox.com`.

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

[3] Universal, 64/32-bit, 66/33MHz, Myrinet/PCI interfaces. http://www.myri.com/news/99903/index.html.

[4] Infiniband Trade Association. http://www.infinibandta.org.

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

[6] P. Balaji. *High Performance Communication Support for Sockets-based Applications over High-speed Networks*. PhD thesis, The Ohio State University, Columbus, OH, 2006.

[7] 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 *the Workshop on Communication Architecture for Clusters (CAC); held in conjunction with the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Rhodes Island, Greece, Apr 2006.

[8] P. Balaji, H. W. Jin, K. Vaidyanathan, and D. K. Panda. Supporting iWARP Compatibility and Features for Regular Network Adapters. In *Workshop on Remote Direct Memory Access (RDMA): Applications Implementations, and Technologies (RAIT); held in conjunction with IEEE International Conference on Cluster Computing*, 2005.

[9] P. Balaji, S. Narravula, K. Vaidyanathan, H. W. Jin, and Dhabaleswar K. Panda. On the Provision of Prioritization and Soft QoS in Dynamically Reconfigurable Shared Data-Centers over InfiniBand. In *the Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2005.

[10] 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 *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2004.

[11] P. Balaji, H. V. Shah, and D. K. Panda. Sockets vs RDMA Interface over 10-Gigabit Networks: An In-depth analysis of the Memory Traffic Bottleneck. In *Workshop on Remote Direct Memory Access (RDMA): Applications, Implementations, and Technologies (RAIT); held in conjunction with IEEE International Conference on Cluster Computing*, San Diego, CA, Sep 20 2004.

[12] P. Balaji, P. Shivam, P. Wyckoff, and D.K. Panda. High Performance User Level Sockets over Gigabit Ethernet. In *Proceedings of the IEEE International Conference on Cluster Computing*, September 2002.

[13] P. Balaji, K. Vaidyanathan, S. Narravula, H.-W. Jin K. Savitha, and D.K. Panda. Exploiting Remote Memory Operations to Design Efficient Reconfiguration for Shared Data-Centers over InfiniBand. In *Proceedings of Workshop on Remote Direct Memory Access (RDMA): Applications, Implementations, and Technologies (RAIT 2004); held in conjunction with the IEEE International Conference on Cluster Computing*, San Diego, CA, September 2004.

[14] P. Balaji, J. Wu, T. Kurc, U. Catalyurek, D. K. Panda, and J. Saltz. Impact of High Performance Sockets on Data Intensive Applications. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC)*, 2003.

[15] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. K. Su. Myrinet: A Gigabit-per-Second Local Area Network. http://www.myricom.com.

[16] A. Cohen, S. Rangarajan, and H. Slye. On the Performance of TCP Splicing for URL-aware Redirection. In *the Proceedings of the USENIX Symposium on Internet Technologies and Systems*, October 1999.

[17] Remote Direct Memory Access Consortium. http://www.rdmaconsortium.org.

[18] Myricom Corporations. The GM Message Passing System.

[19] P. Culley, U. Elzur, R. Recio, and S. Bailey. Marker PDU Aligned Framing for TCP Specification, November 2002.

[20] W. Feng, P. Balaji, C. Baron, L. N. Bhuyan, and D. K. Panda. Performance Characterization of a 10-Gigabit Ethernet TOE. In *Proceedings of Hot Interconnects Symposium*, 2005.

[21] W. Feng, J. Hurwitz, H. Newman, S. Ravot, L. Cottrell, O. Martin, F. Coccetti, C. Jin, D. Wei, and S. Low. Optimizing 10-Gigabit Ethernet for Networks of Workstations, Clusters and Grids: A Case Study. In *Proceedings of the Supercomputing (SC)*, 2003.

[22] Internet Engineering Task Force. http://www.ietf.org.

[23] H. Frazier and H. Johnson. Gigabit Ethernet: From 100 to 1000Mbps.

[24] D. Goldenberg, M. Kagan, R. Ravid, and M. Tsirkin. Transparently Achieving Superior Socket Performance using Zero Copy Socket Direct Protocol over 20 Gb/s InfiniBand Links. In *Workshop on Remote Direct Memory Access (RDMA): Applications Implementations, and Technologies (RAIT); held in conjunction with IEEE International Conference on Cluster Computing*, 2005.

[25] D. Goldenberg, M. Kagan, R. Ravid, and M. Tsirkin. Zero Copy Sockets Direct Protocol over InfiniBand - Preliminary Implementation and Performance Analysis. In *IEEE Hot Interconnects: A Symposium on High Performance Interconnects*, 2005.

[26] http://www.top500.org. Top 500 supercomputer sites.

[27] J. Hurwitz and W. Feng. Initial End-to-End Performance Evaluation of 10-Gigabit Ethernet. In *IEEE Hot Interconnects: A Symposium on High-Performance Interconnects*, Palo Alto, California, 2003.

[28] J. Hurwitz and W. Feng. End-to-End Performance of 10-Gigabit Ethernet on Commodity Systems. *IEEE Micro*, 2004.

[29] H. W. Jin, S. Narravula, G. Brown, K. Vaidyanathan, P. Balaji, and D. K. Panda. Performance Evaluation of RDMA over IP: A Case Study with the Ammasso Gigabit Ethernet NIC. In *Workshop on High Performance Interconnects for Distributed Computing (HPI-DC); In conjunction with HPDC-14*, 2005.

[30] J. S. Kim, K. Kim, and S. I. Jung. SOVIA: A User-level Sockets Layer over Virtual Interface Architecture. In *Proceedings of the International Conference on Cluster Computing*, 2001.

[31] S. Narravula, P. Balaji, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Supporting Strong Coherency for Active Caches in Multi-Tier Data-Centers over InfiniBand. In *SAN-1 Workshop, held in conjunction with Int'l Symposium on High Performance Computer Architecture (HPCA-8)*, 2004.

[32] Network-Based Computing Laboratory. MVAPICH: MPI over InfiniBand. http://nowlab.cse.ohio-state.edu/projects/mpi-iba/index.html.

[33] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM). In *Proceedings of Supercomputing (SC)*, 1995.

[34] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner. Internet Small Computer Systems Interface (iSCSI), RFC 3720, April 2004.

[35] H. V. Shah, C. Pu, and R. S. Madukkarumukumana. High Performance Sockets and RPC over Virtual Interface (VI) Architecture. In *International Workshop on Communication and Architectural Support for Network-Based Parallel Computing (CANPC)*, 1999.

[36] P. Shivam, P. Wyckoff, and D. K. Panda. Can User-Level protocols take advantage of Multi-CPU NICs? In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2002.

[37] P. Shivam, P. Wyckoff, and D.K. Panda. EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing. In *Proceedings of Supercomputing (SC)*, 2001.

[38] W. R. Stevens. *TCP/IP Illustrated, Volume I: The Protocols.* Addison Wesley, 2nd edition, 2000.

[39] J. Stone and C. Partridge. When the CRC and TCP Checksum Disagree. In *ACM SIGCOMM*, 2000.

[40] S. Sur, L. Chai, H.-W. Jin, and D. K. Panda. Shared Receive Queue based Scalable MPI Design for InfiniBand Clusters. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[41] S. Sur, M. J. Koop, and Dhabaleswar K. Panda. High-Performance and Scalable MPI over InfiniBand With Reduced Memory Usage: An In-Depth Performance Analysis. In *Proceedings of SuperComputing*, 2006.

[42] G. R. Wright and W. R. Stevens. *TCP/IP Illustrated, Volume II: The Implementation.* Addison Wesley, 2nd edition, 2000.