iWarp-Protocol Kernel-Space Software Implementation

Dennis Dalessandro, Ananth Devulapalli, Pete Wyckoff

{dennis, ananth, pw}@osc.edu

Ohio Supercomputer Center

Overview

- Introduction
- Motivation: *Why Software iWarp?*
- iWarp: Details
- Implementation details
- Experiments & Results
- Our future goals

Introduction

- High Performance Interconnects
  - Zero-copy
  - RDMA
  - Specialty protocol
  - LAN-wide
- RDMA over Ethernet $\longrightarrow$ *iWarp*
  - De-congest data-path at the end-points
  - 10 GBps at 3-4 GHz

Motivation

- Single-sided Acceleration
- Flexible Research Platform
- Advantages of iWarp in kernel

Motivation

- Single-sided Acceleration
  - Hardware-enabled Server, Software-enabled Clients
  - Performance penalty at software end ☹
  - Hardware Accelerated server ☺
  - Cost-effective intermediate step
- Flexible Research Platform
- Advantages of iWarp in kernel

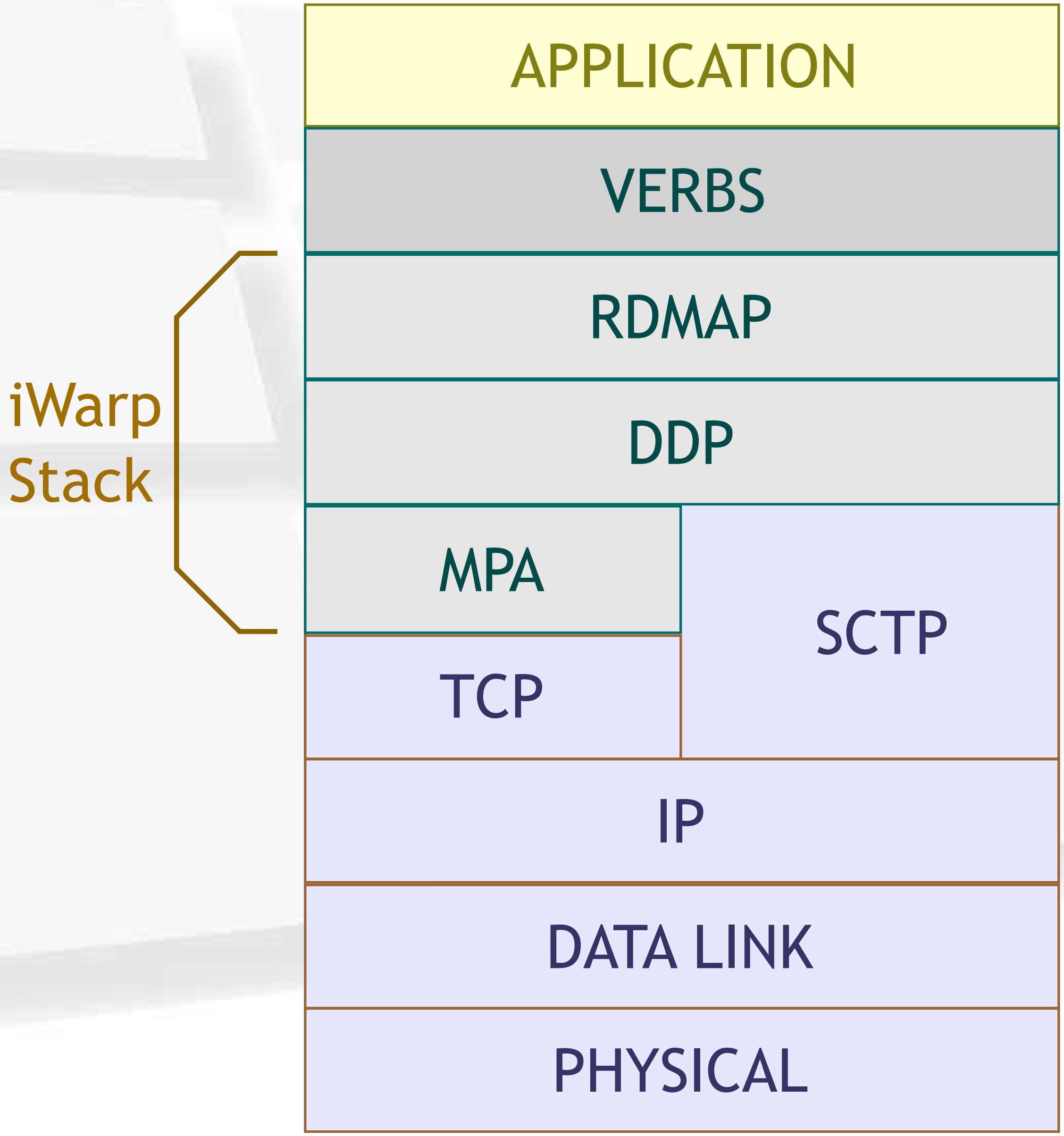
Motivation

- Single-sided Acceleration
  - Hardware-enabled Server, Software-enabled Clients
  - Performance penalty at software end ☹
  - Hardware Accelerated server ☺
  - Cost-effective intermediate step
- Flexible Research Platform
  - Protocol Experimentation
  - Protocol Compliance
  - Extensible to other protocols: *iSER, SRP*
- Advantages of iWarp in kernel

Motivation

- Single-sided Acceleration
  - Hardware-enabled Server, Software-enabled Clients
  - Performance penalty at software end ☹
  - Hardware Accelerated server ☺
  - Cost-effective intermediate step
- Flexible Research Platform
  - Protocol Experimentation
  - Protocol Compliance
  - Extensible to other protocols: *iSER, SRP*
- Advantages of iWarp in kernel
  - Unlock iWarp for kernel-resident clients: *NFS*
  - Coupling with TCP
  - Reduction in overhead

iWarp Details



Implementation Issues

- Verbs
- TCP Interface
- Threading Model
- Memory Registration Issues

Impl. Issues: *Verbs*

- Verbs or API like DAPL?
- User-space resident
- Character device interface with kernel module
- Modularized implementation
  - Single code-base for both user and kernel based implementations
- Minimize scope without sacrificing functionality

Impl. Issues: *TCP Interface*

- `kernel_sendmsg`, `kernel_recvmsg`
  - Blocked sends
  - Polling recvs
- MPA loosely coupled with TCP
  - *Flexibility versus Functionality*

Impl. Issues: *Threading Model*

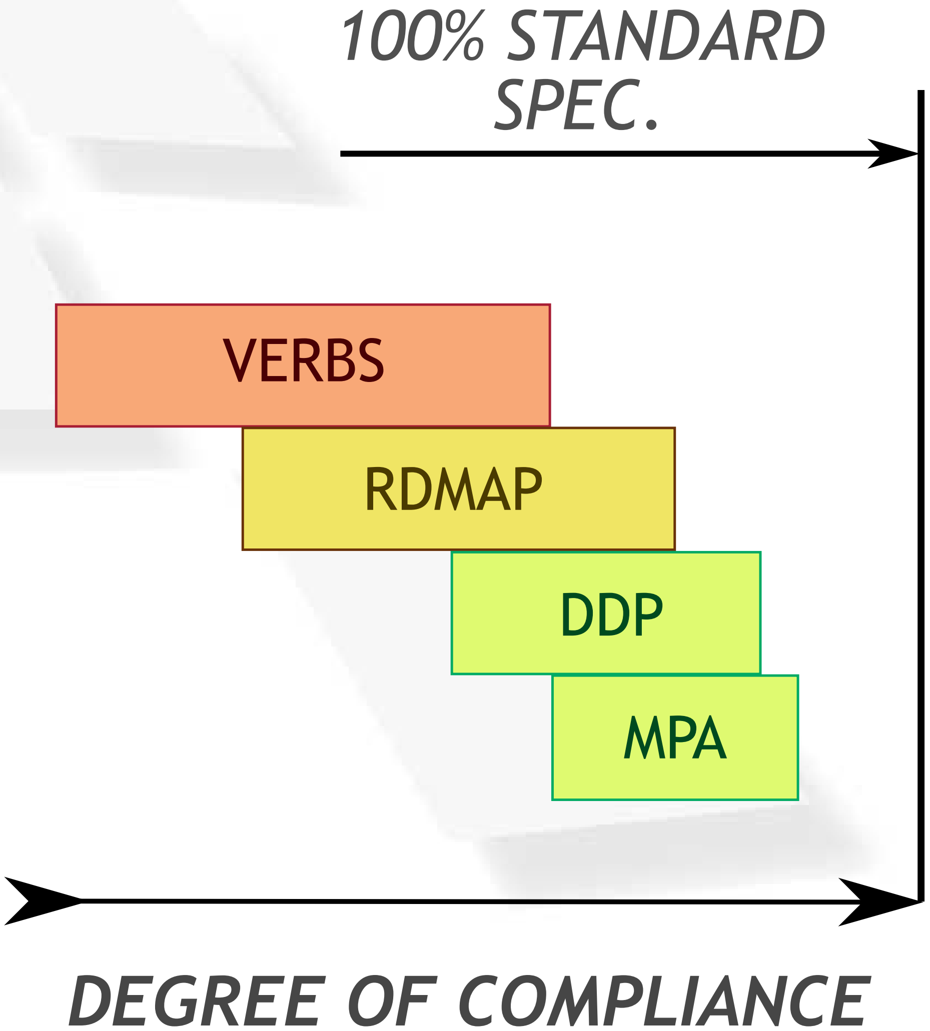
- Multi-threading: non-blocking, asynchronous
- Single Threaded model
  - Simplicity versus Performance

Impl. Issues: *Memory Registration*

- Pre-registration of application buffers
- `kmap` and `kunmap`
- Book-keeping using reference counting
- 64-bit machines
- Overhead

About the code base

- iWarp software stack works against Ammasso 1100 RNIC
- CRC and Markers: switched on and off
- 20,000 lines of ANSI C code (user and kernel)
- Linux 2.6 kernel
- 32-bit and 64-bit support



Experimental Setup

- 71 node cluster with 41 Ammasso 1100 RNIC cards
  - Beta cards with FPGA-based IP
  - RDMA data-path and TCP data-path
- Dual Opteron 250 processors
  - One processor disabled for utilization tests
- 2GB RAM, 80GB SATA drives
- 2 Tigon Gigabit Ethernet NICs
- Tyan S2891 Motherboard
- 2 SMC switches
  - Switches introduce 2.8 μs latency

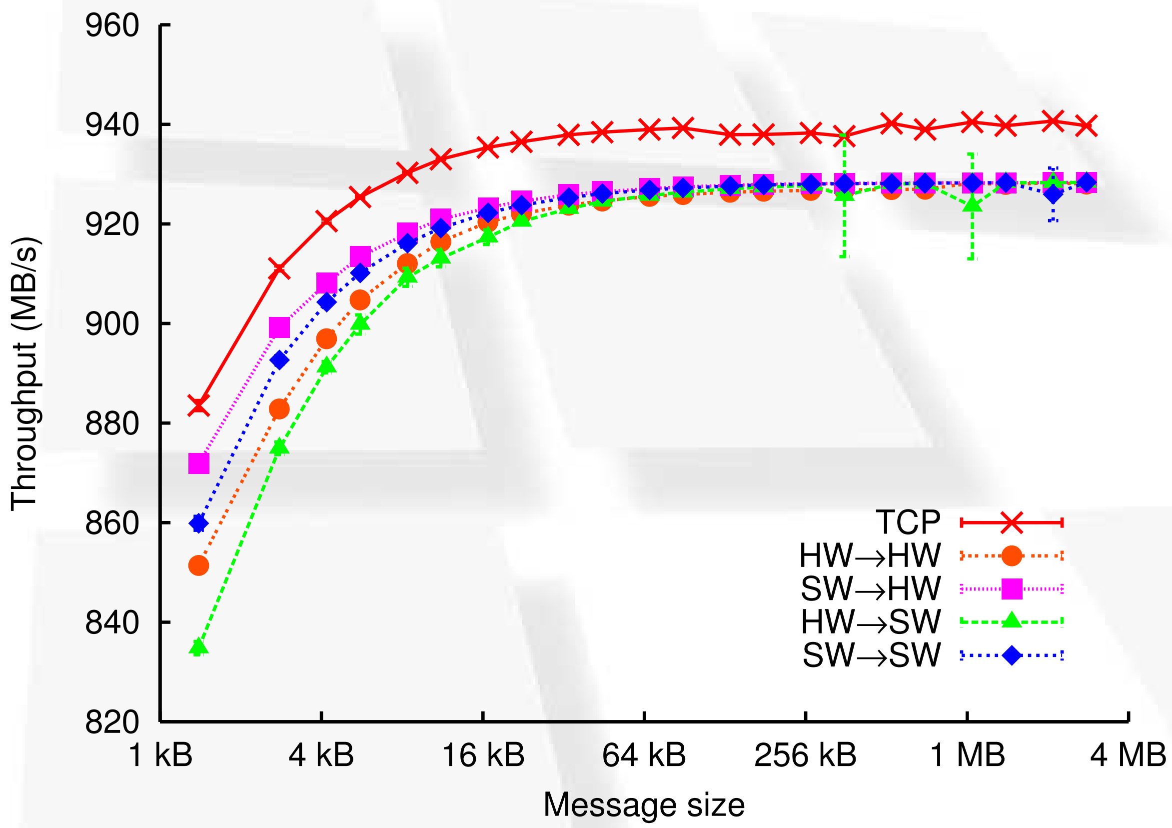
Latency

	4 byte messages	64 kB messages
hw-hw	16.1 ± 0.3	614.2 ± 3.3
ksw-hw	18.7 ± 0.2	619.7 ± 1.2
tcp-tcp	16.9 ± 0.2	594.8 ± 18.9

Table 1: Latency overview (μs).

- Latency: ½-way ping-pong delay
- Back-to-back: bypass switch
- Small overhead

Throughput



- Sender and Receiver
- TCP > 10 MBps

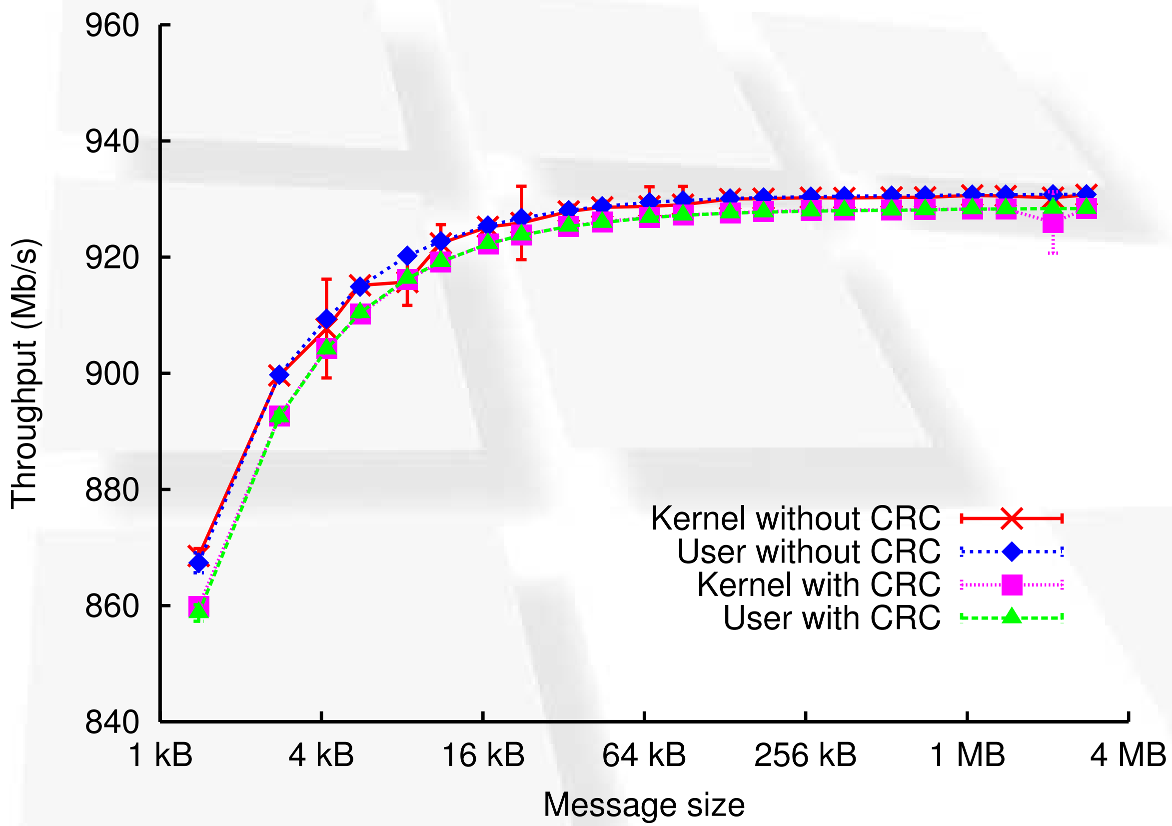
Latency: *Kernel v/s User*

	4 B messages	64 kB messages
kernel with CRC	20.3 ± 0.2	615.5 ± 1.2
user with CRC	19.6 ± 0.2	612.3 ± 1.9
kernel without CRC	20.1 ± 0.2	604.5 ± 0.8
user without CRC	19.5 ± 0.2	602.7 ± 0.8

Table 2: User vs kernel space latency (μs).

- CQ in kernel
- `kmap/kunmap` overhead

Throughput: *Kernel v/s User*



- CRC takes away 8 MBps
- Kernel and User space similar

CPU utilization Hardware

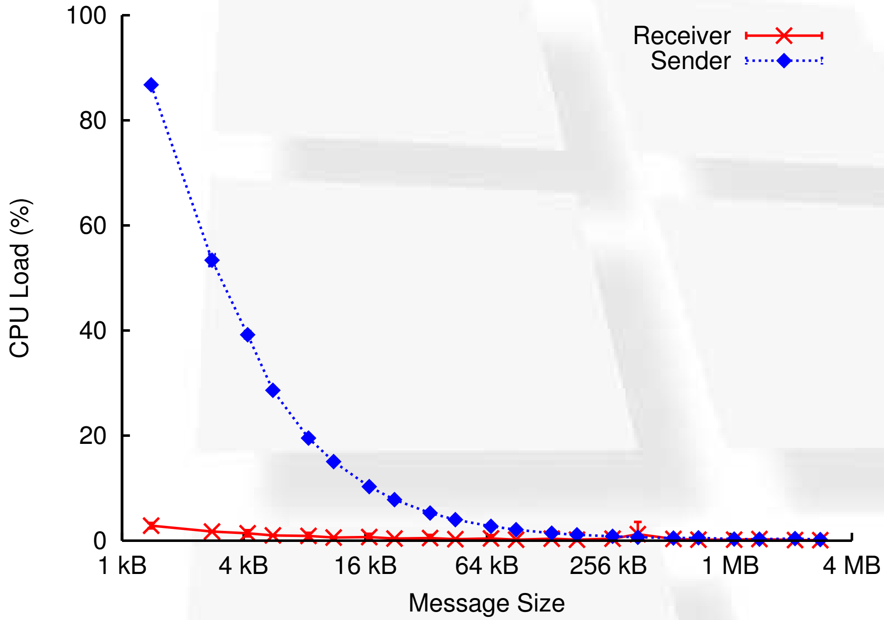


Figure 1: Hardware ⇔ Hardware

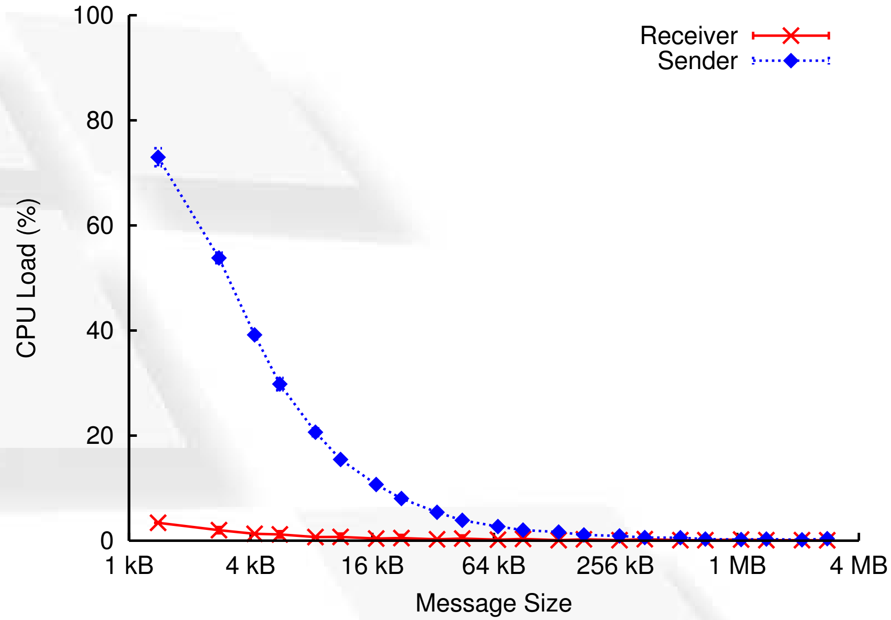


Figure 2: Hardware ⇔ Software

- Subtractive method
- Hardware and Software Identical

CPU utilization TCP and Software

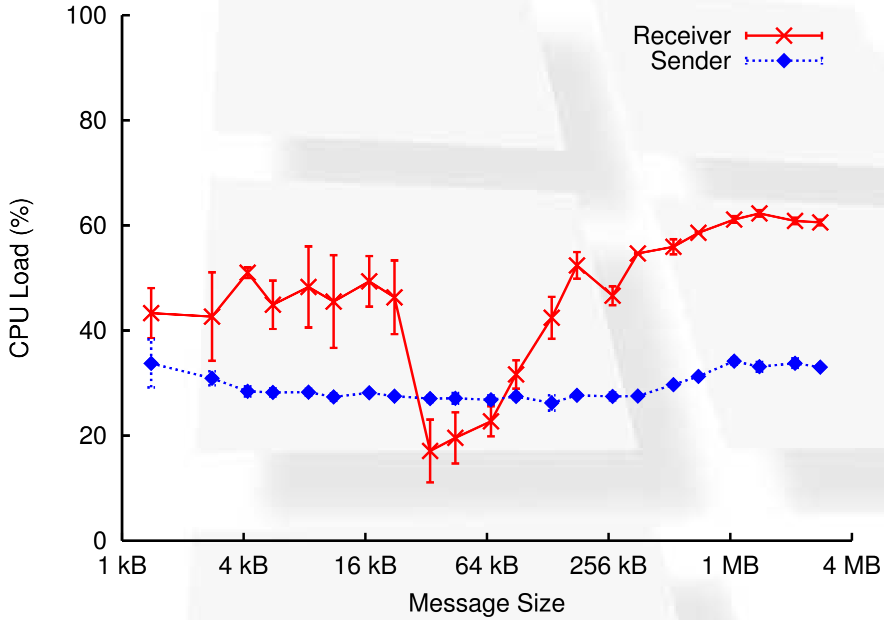


Figure 3: TCP ⇔ TCP

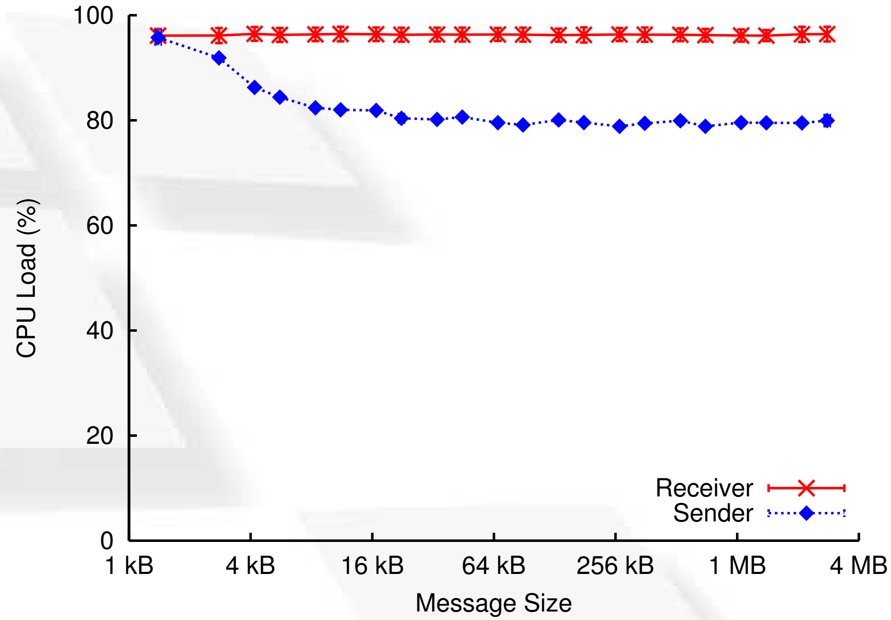
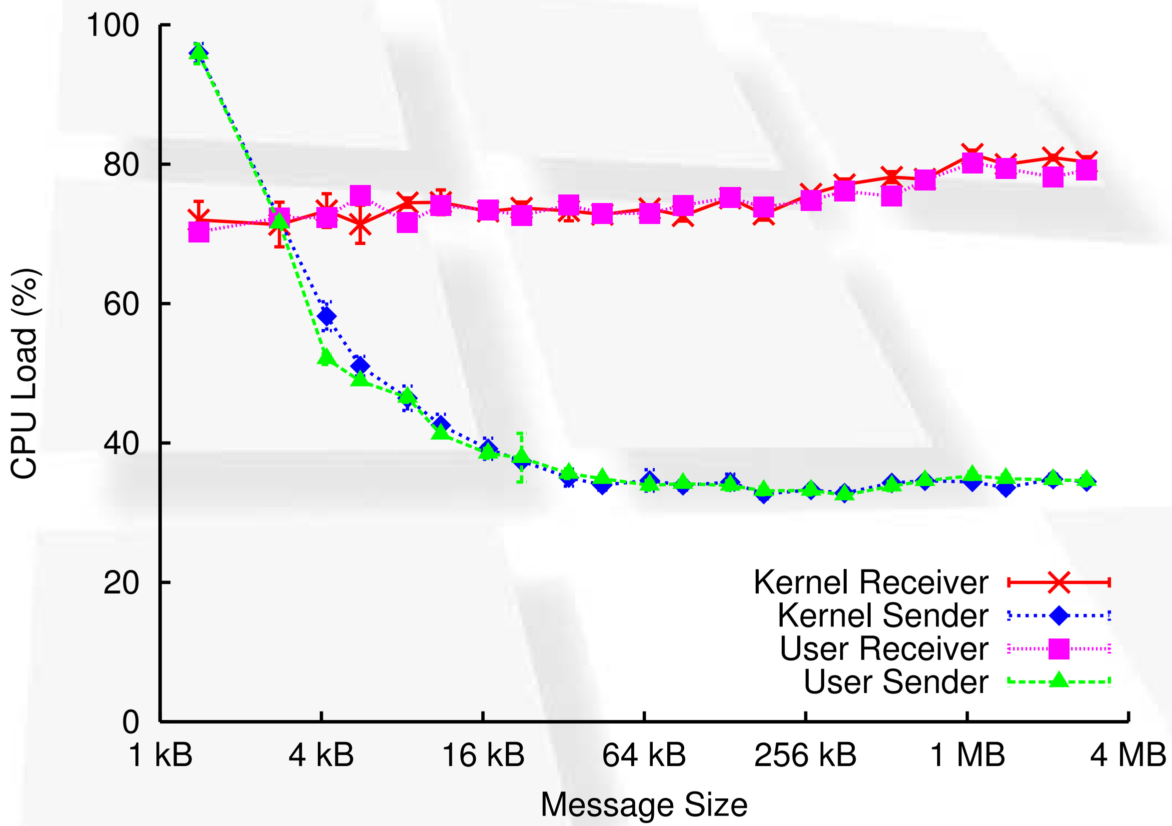


Figure 4: Software ⇔ Software

- TCP costly than Hardware iWarp
- Software is CPU intensive: *CRC*

CPU Utilization without CRC



- 20% for Reciever and 40% Sender load due to CRC
- The loads in range of TCP

Related Work

- User-space software iWarp
- Sockets-based iWarp
- Other verbs: DAT Collaborative, OpenFabrics

Future Work

- Porting kernel space clients
- Integrating MPA with TCP
- WAN deployment
- iSER/SRP extensions
- Multithreaded stack

Conclusions

- Demonstrated interoperability with Hardware iWarp
- Demonstrated single-sided acceleration capability
- Software iWarp for kernel-resident clients
- Software iWarp is logical step before full deployment

Software Availability

http://www.osc.edu/research/network_file/projects/iwarp/index.shtml