

# Performance Issues in Parallelized Network Protocols

Erich M. Nahum David J. Yates James F. Kurose  
Don Towsley



University of Massachusetts at Amherst

## Introduction

Parallelism in network protocols

Shared-memory multiprocessors

High-speed network media

Bandwidth-hungry applications

How to exploit parallelism?

## Differing Approaches to Parallelism

Packet-level parallelism

Connection-level parallelism

Layer parallelism

Functional parallelism

**Our focus: packet-level parallelism**

## Why Packet-Level Parallelism?

No restrictions on packet processing

Natural load balancing

Medium-grained parallelism

Popular form (Goldberg 93, Bjorkman 93)

Appropriate for shared-memory MP's (Schmidt 94)

## Research Issues

Impact of ordering

Locking granularity

Single connection vs. multiple connections

Structuring for cache affinity

Using atomic primitives

Influence of packet size, checksumming

## Experimental Environment

### Hardware and Software

SGI Challenge multiprocessor (8 100 MHz R4400)

Parallelized *x*-kernel

Parallelized TCP, UDP, IP, FDDI stack

In-memory drivers

## Experimental Environment

### Test Methodology

Single-connection throughput

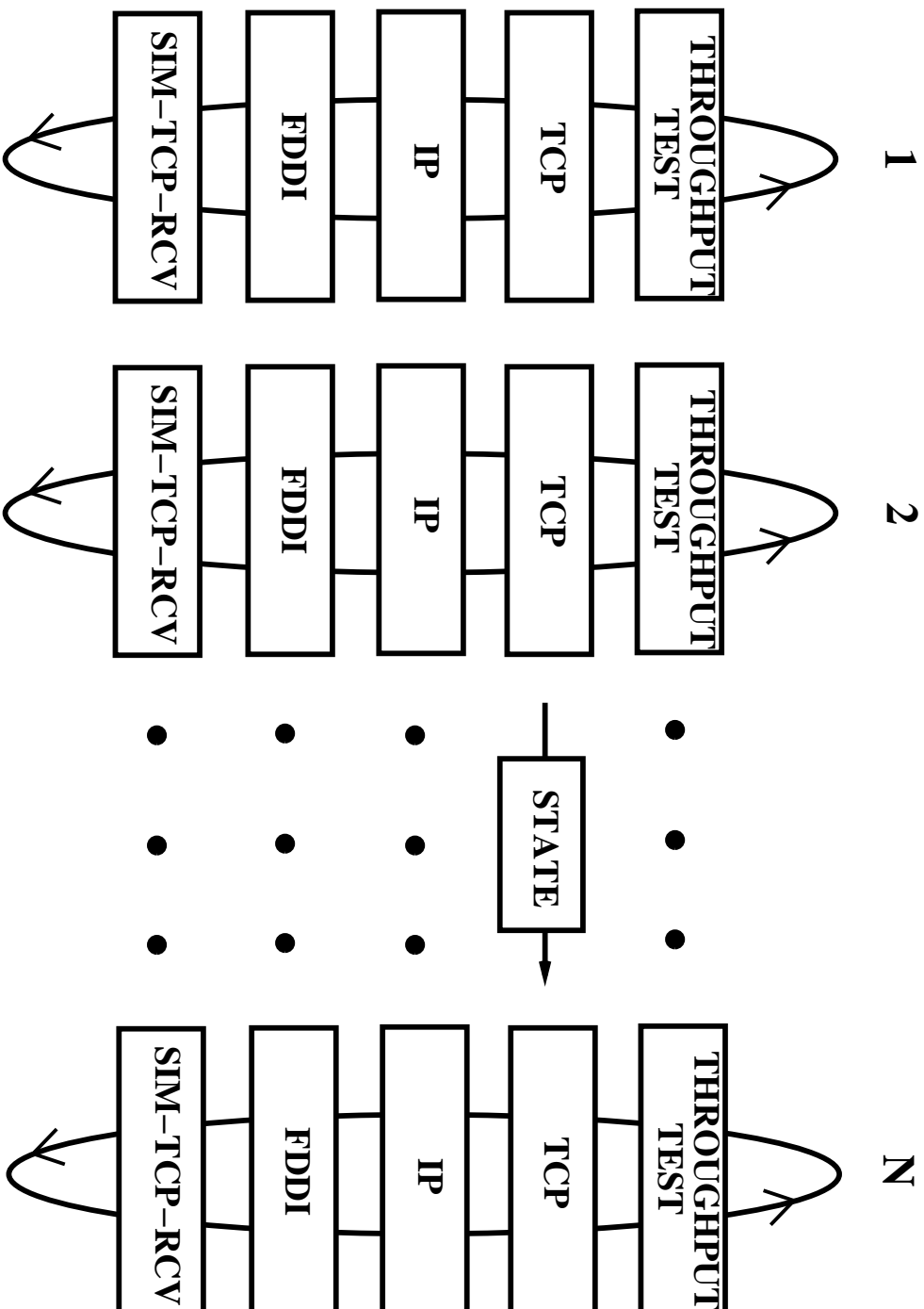
Single multithreaded parallel application

Average of 10 runs with confidence intervals

30 second warmup, 30 second sampling period

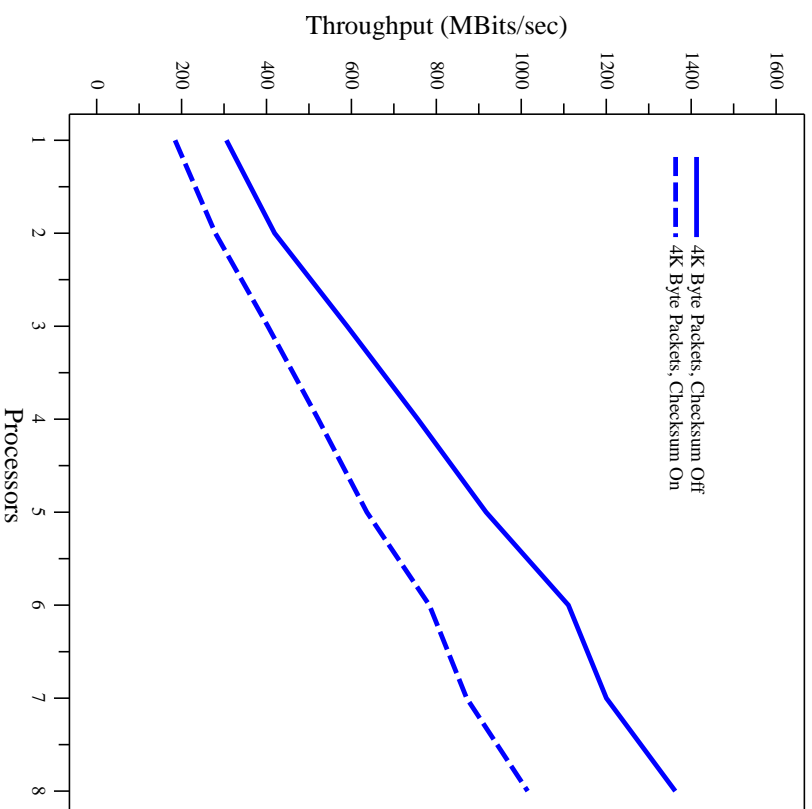
Isolated machine

# Threading Model

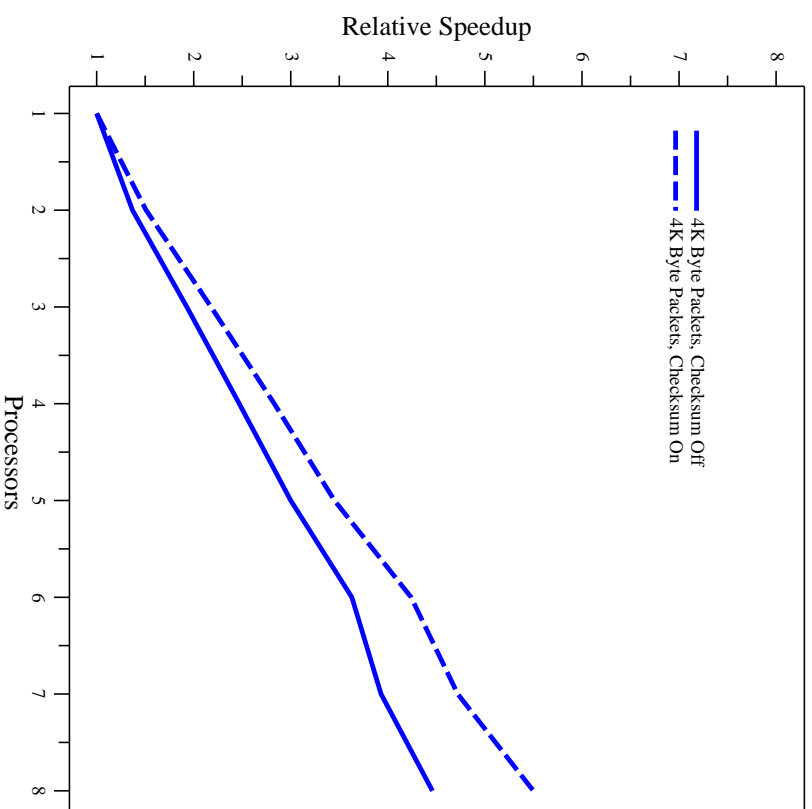


# UDP Send Side

## Throughput



## Speedup



## Locking in TCP

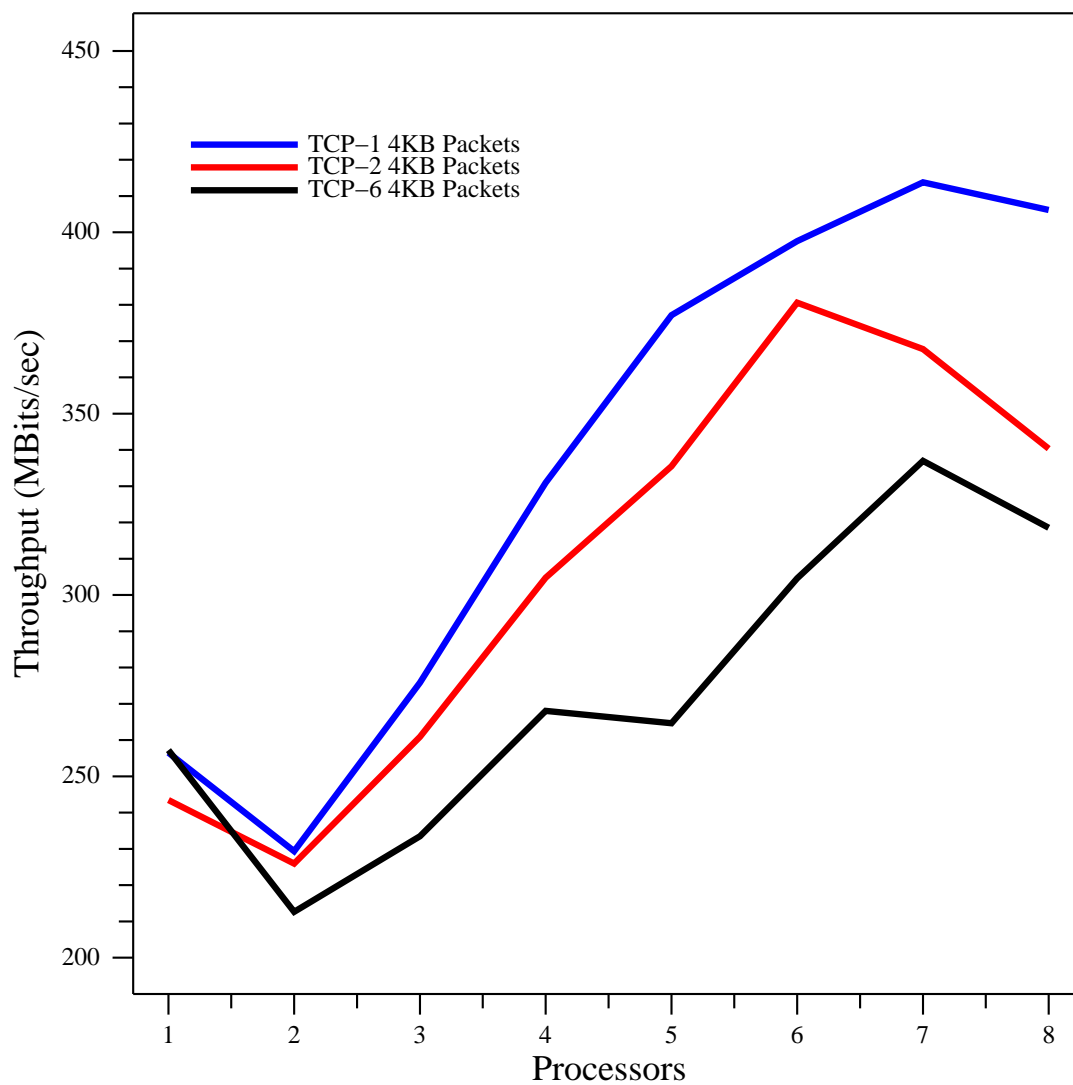
Great deal of TCP connection state

Compare 3 locking strategies:

- SICS style locking, 6 locks for connection state
- 2 locks: send-side state, receive-side state
- 1 lock: all connection state

## Locking in TCP

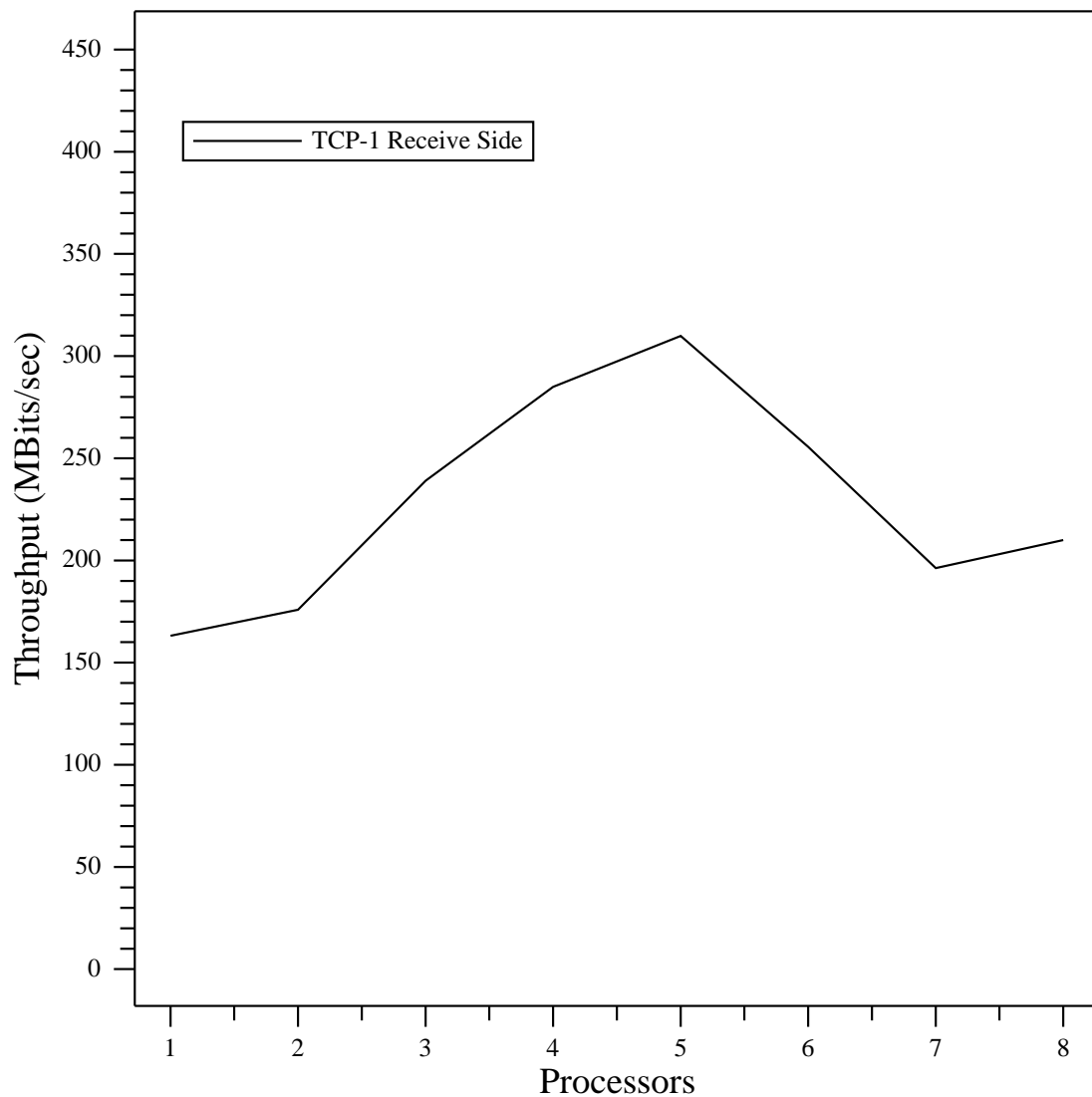
Receive Side, 4 KB Packets, Checksum off



- Single-lock TCP performs best

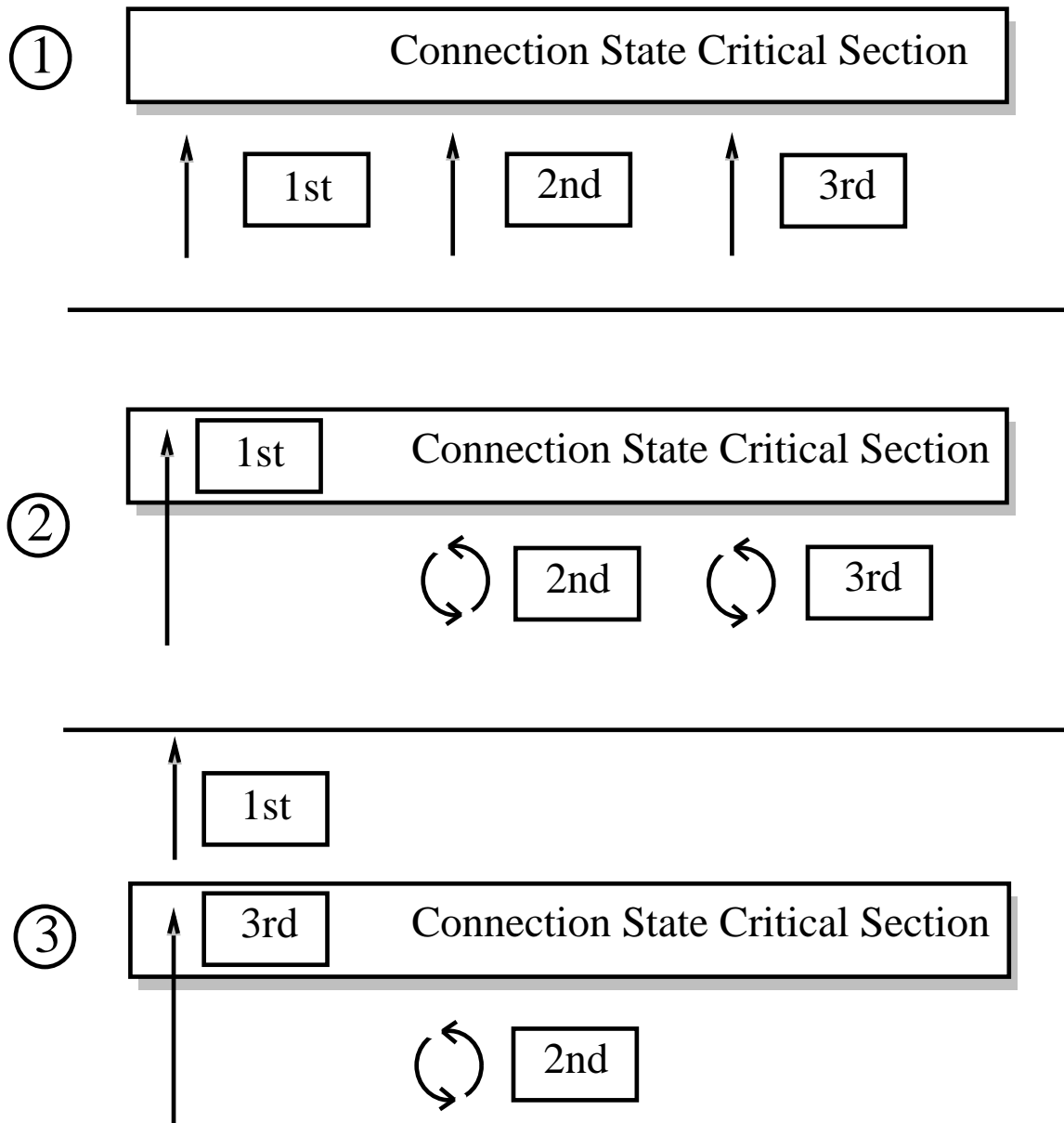
## TCP Receive Side

Recv Side, 4KB Packets, Checksum On



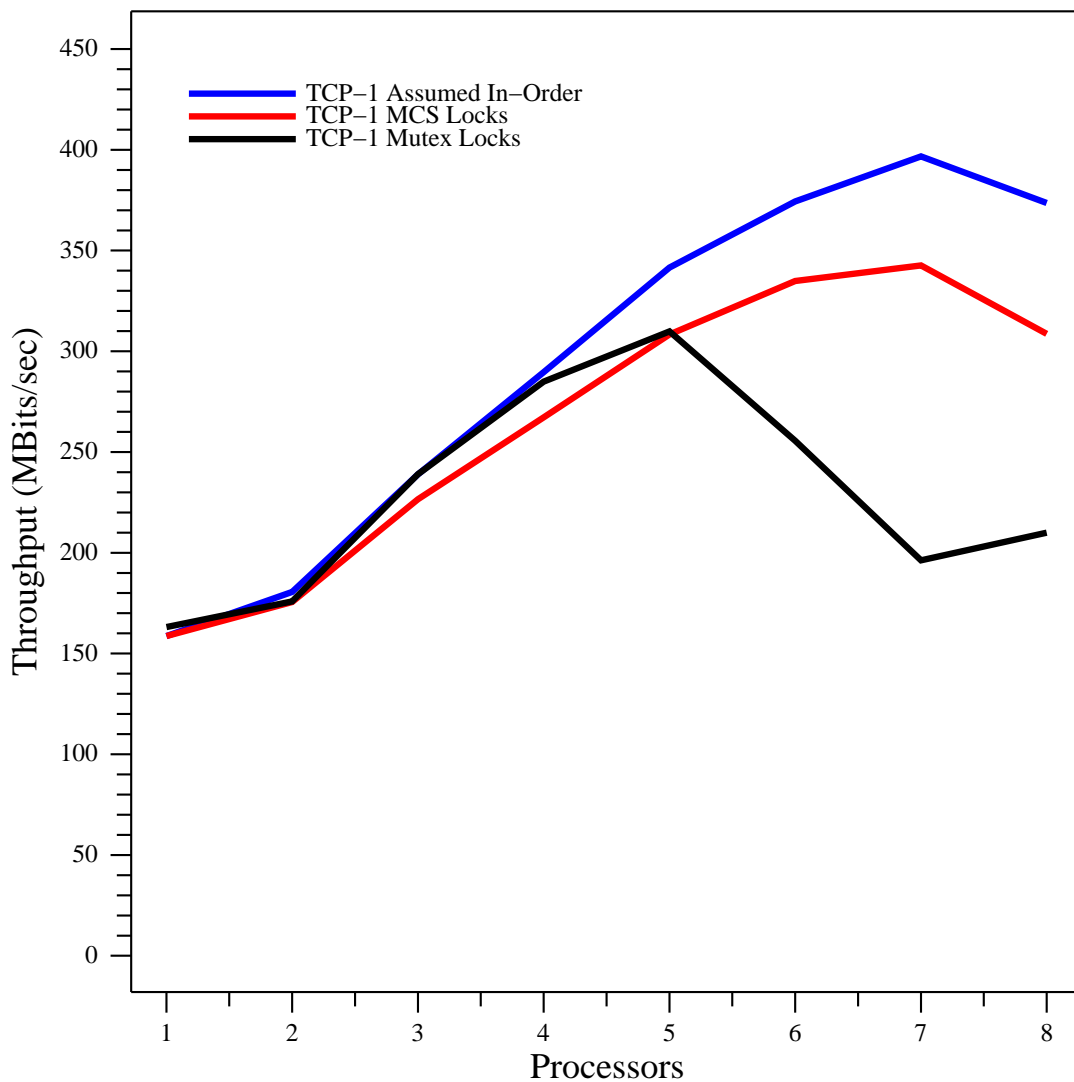
- Why the dip after 5 processors?

# Thread Reordering



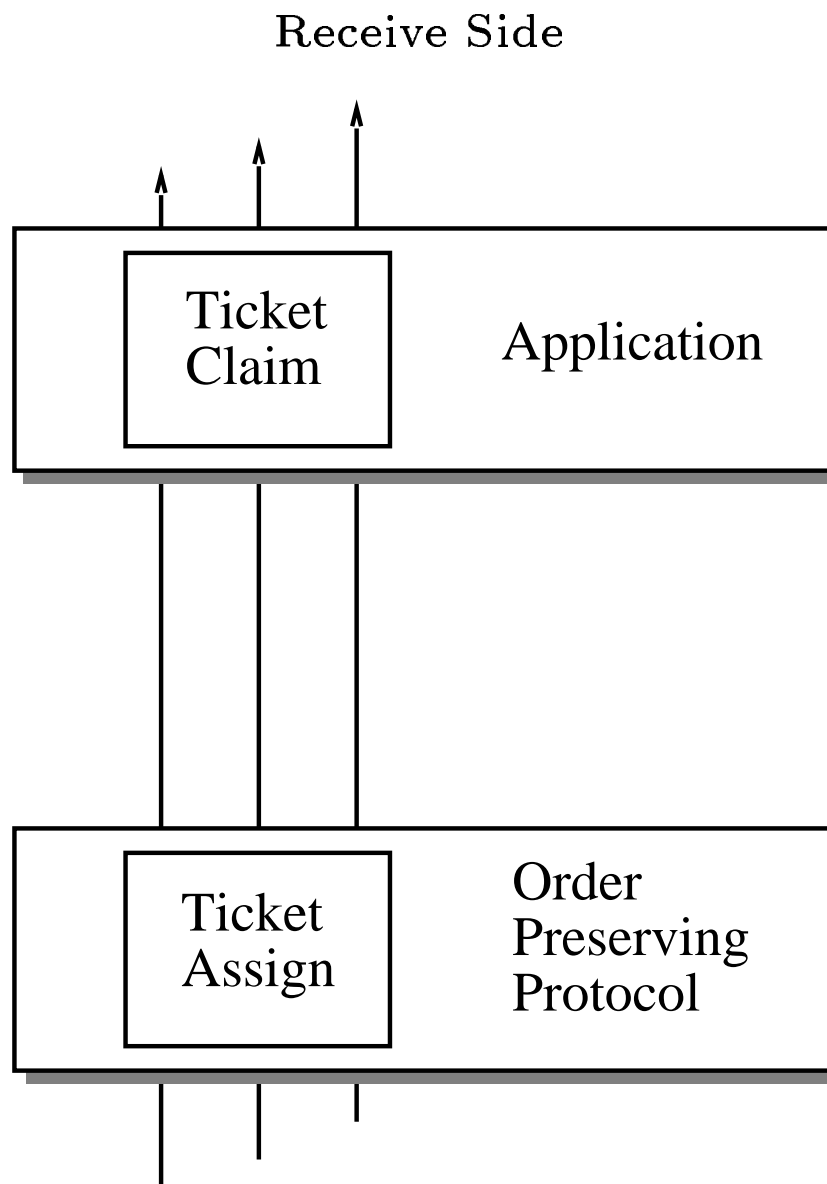
# TCP: Solving the Ordering Problem

Recv Side, 4KB Packets, Checksum On



- MCS queuing locks preserve order

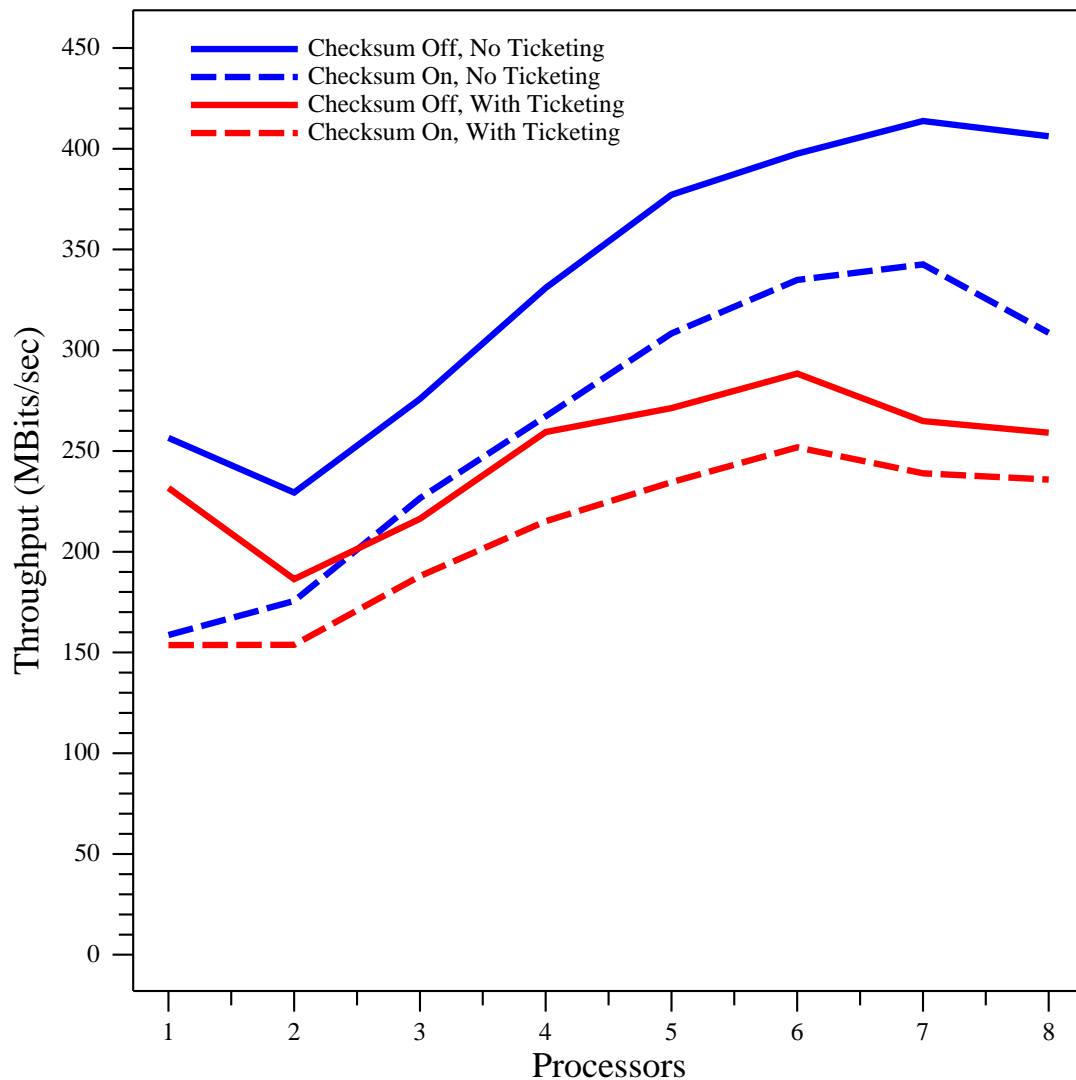
## Ordering to the Application



- Ticketing preserves order

## Ordering to the Application

### Receive Side, 4KB Packets



- Preserving order is expensive

## Multiple Connections

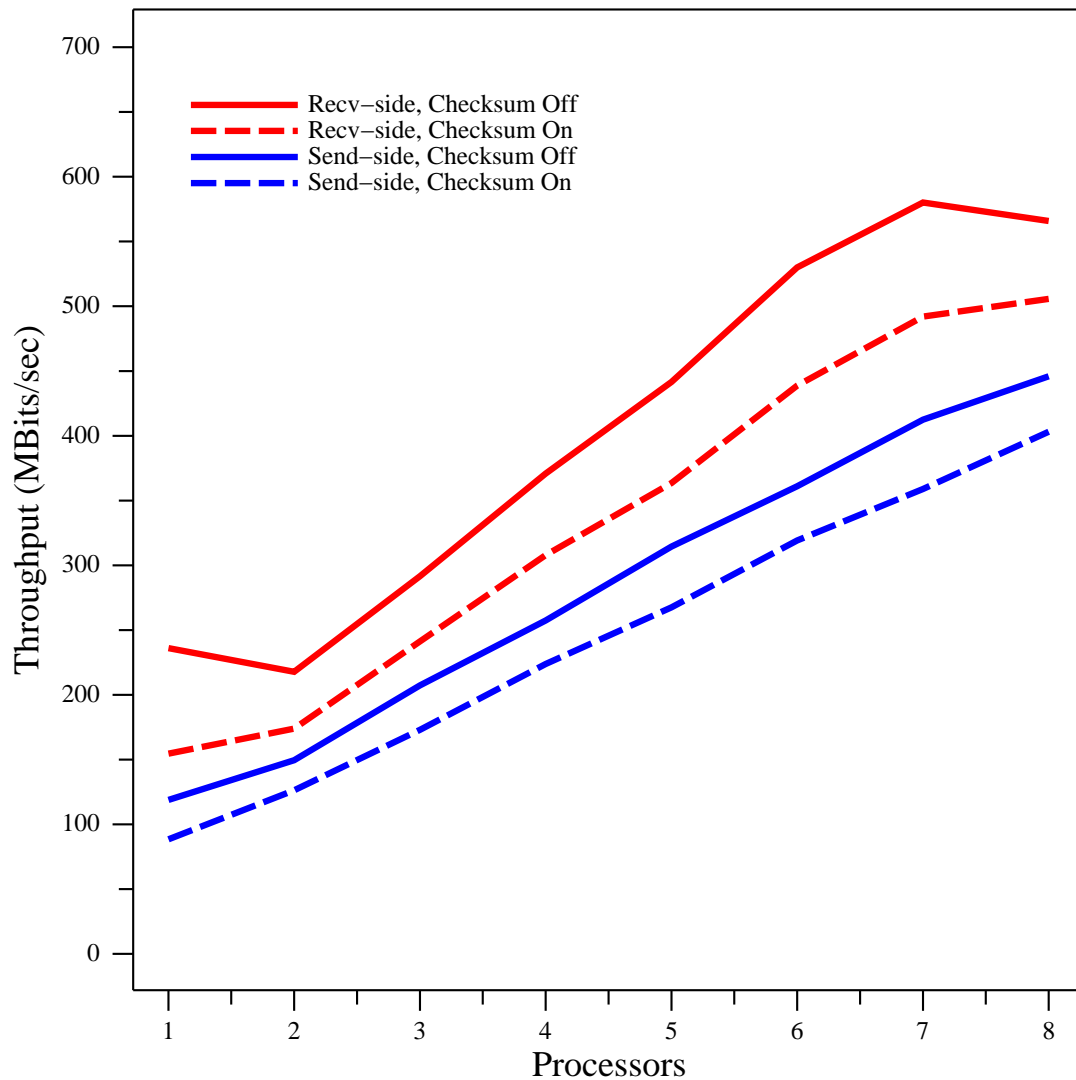
Bottleneck is in TCP connection-state lock

Gain throughput by using multiple connections

Applications using multiple connections must worry about order between connections

## Multiple Connections

Send & Recv, 4KB Packets, Cksum On & Off



- Multiple connections improve scalability

## Other Issues in Paper

*Atomic increment/decrement*

*Managing data for cache affinity*

*Checksumming and packet size*

*Architectural comparisons*

## Conclusions

Ordering is important.

- for performance reasons (TCP)
- for semantic reasons

Simpler locking is better.

Multiple connections improves scalability.

## Future Work

Packet-level parallelism

Larger numbers of processors

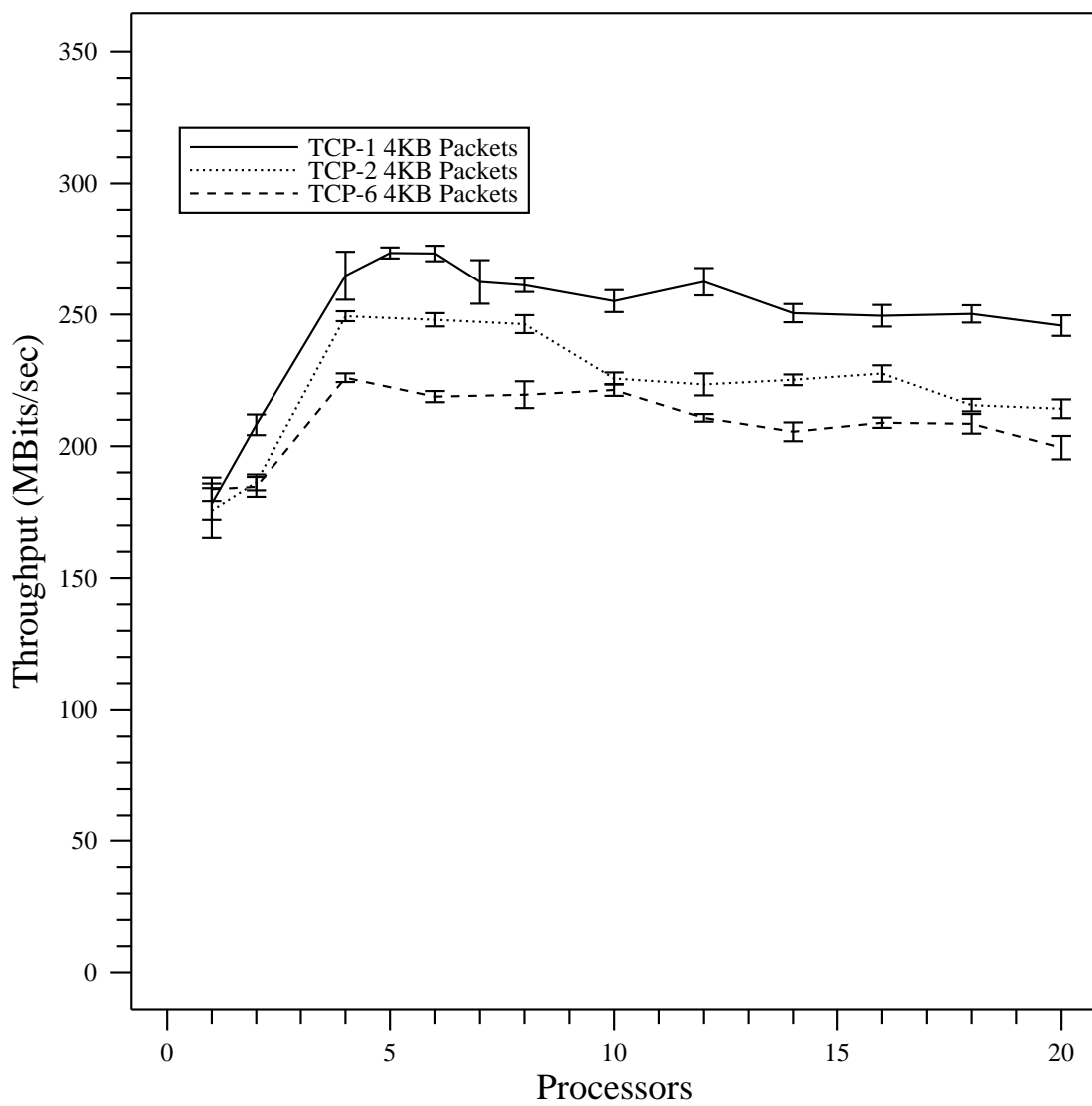
Different types of protocols

More multiple-connection experiments

Connection-level parallelism

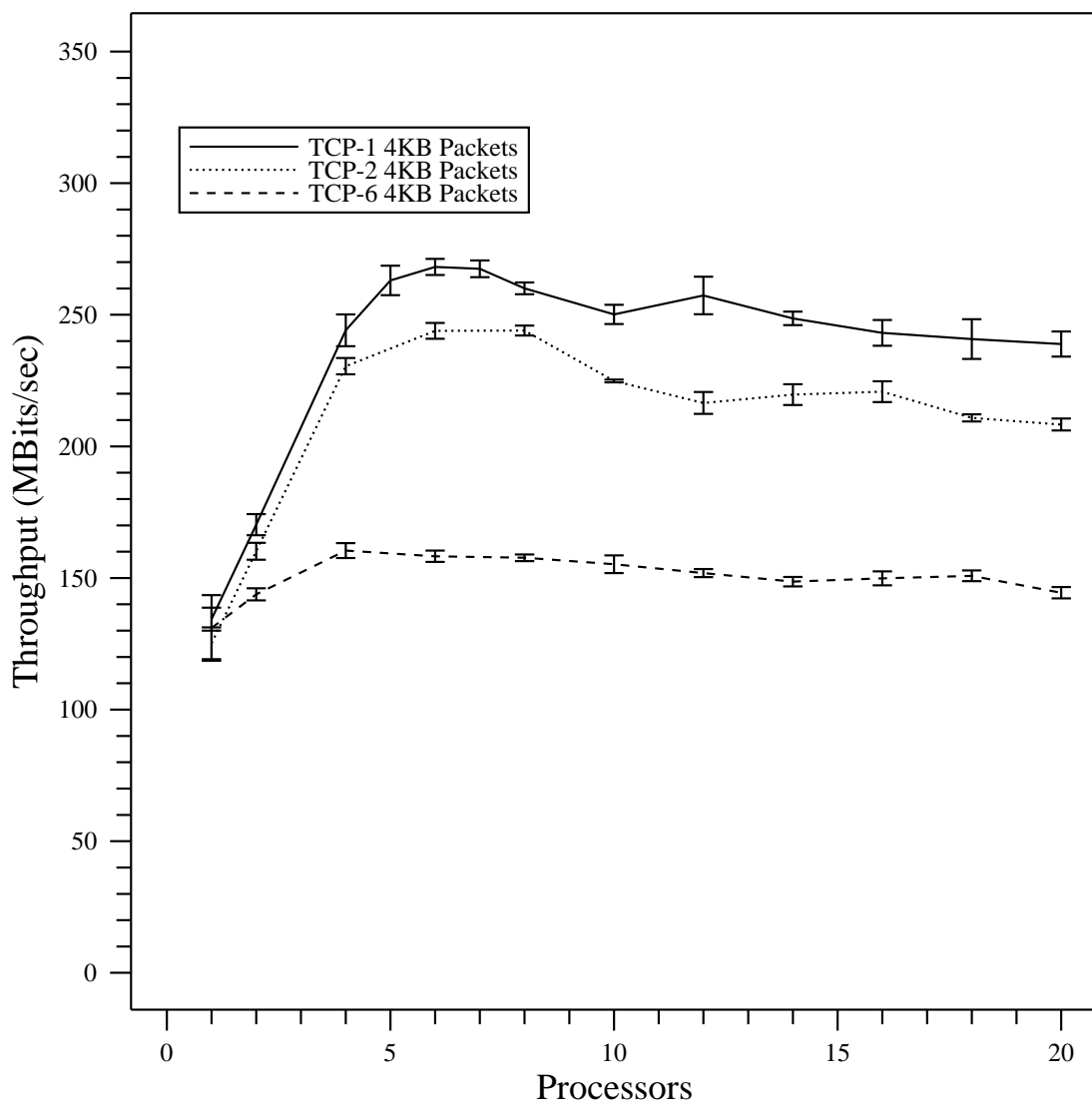
# TCP Send Side

Send Side, 4KB Packets, Checksum Off



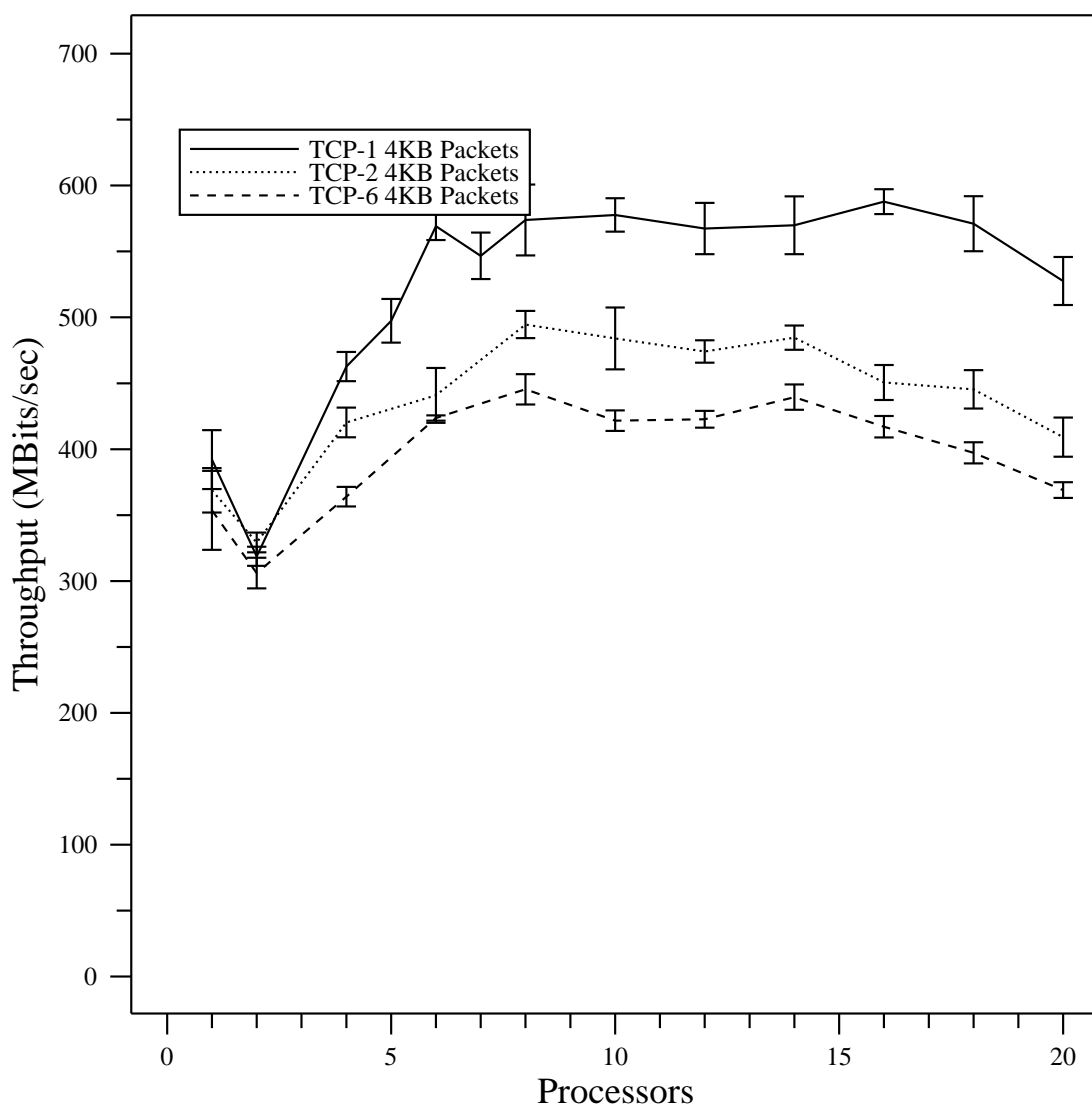
# TCP Send Side

Send Side, 4KB Packets, Checksum On



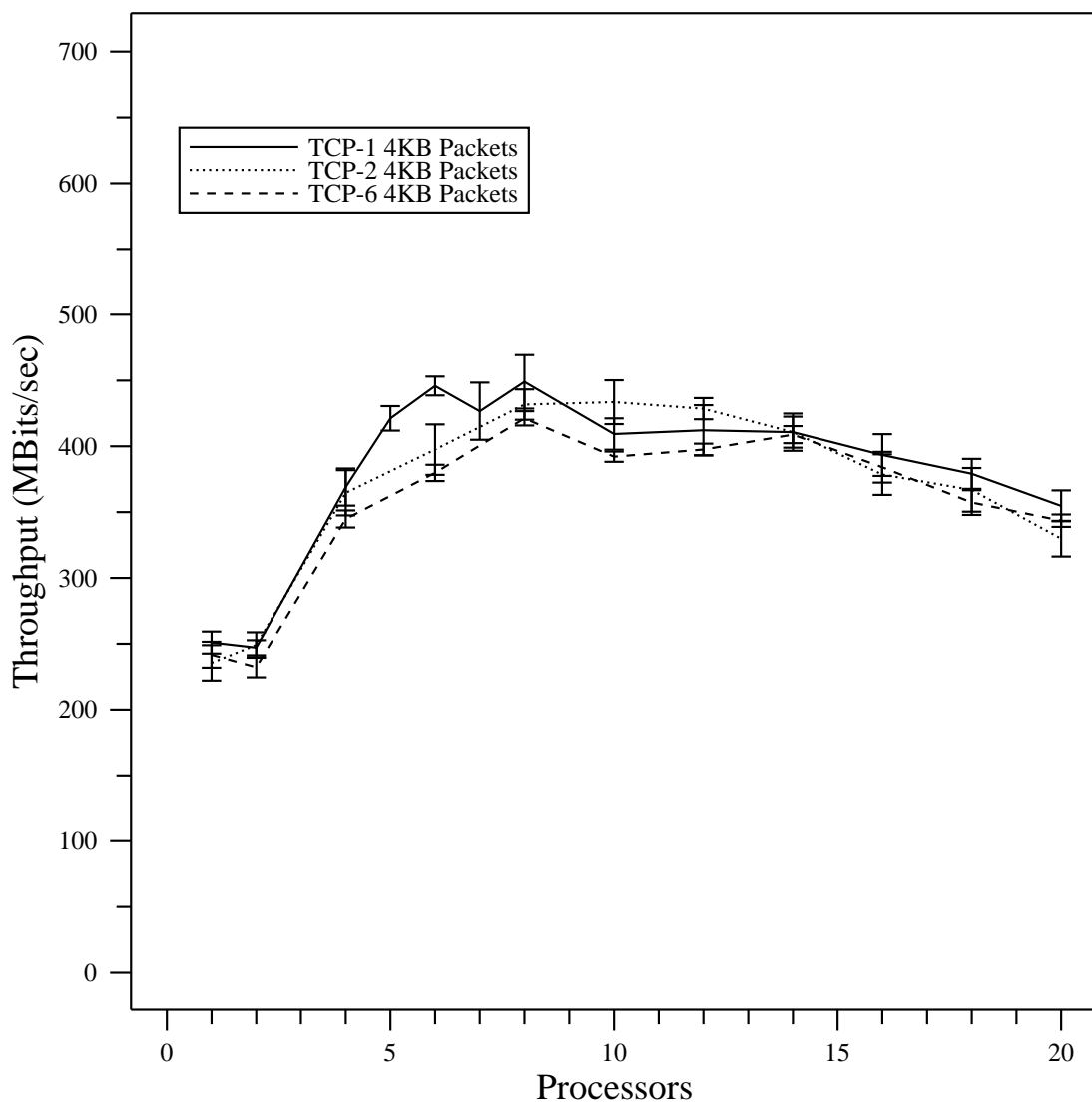
## TCP Receive Side

Recv Side, 4KB Packets, Checksum Off



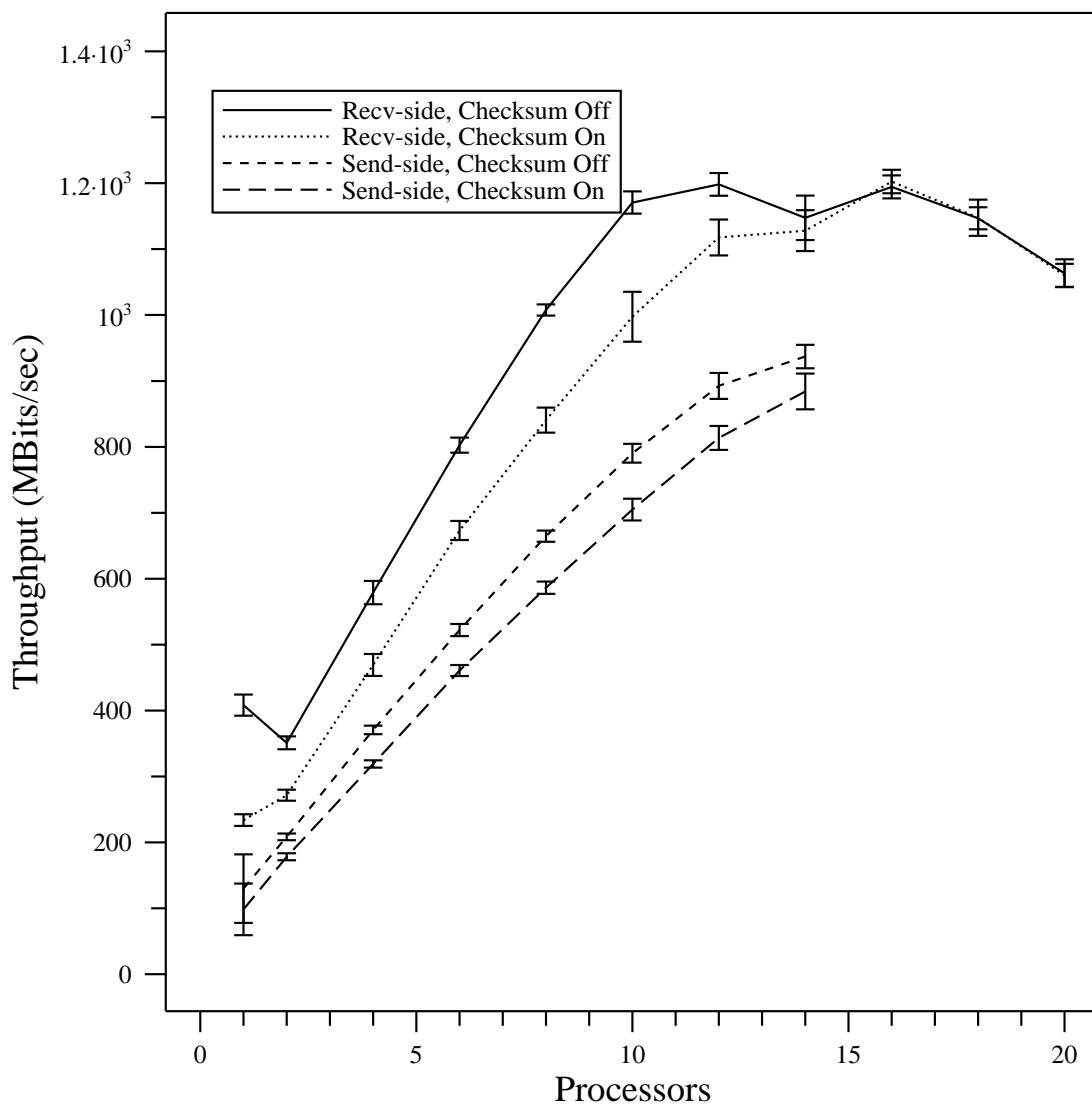
# TCP Receive Side

Recv Side, 4KB Packets, Checksum On



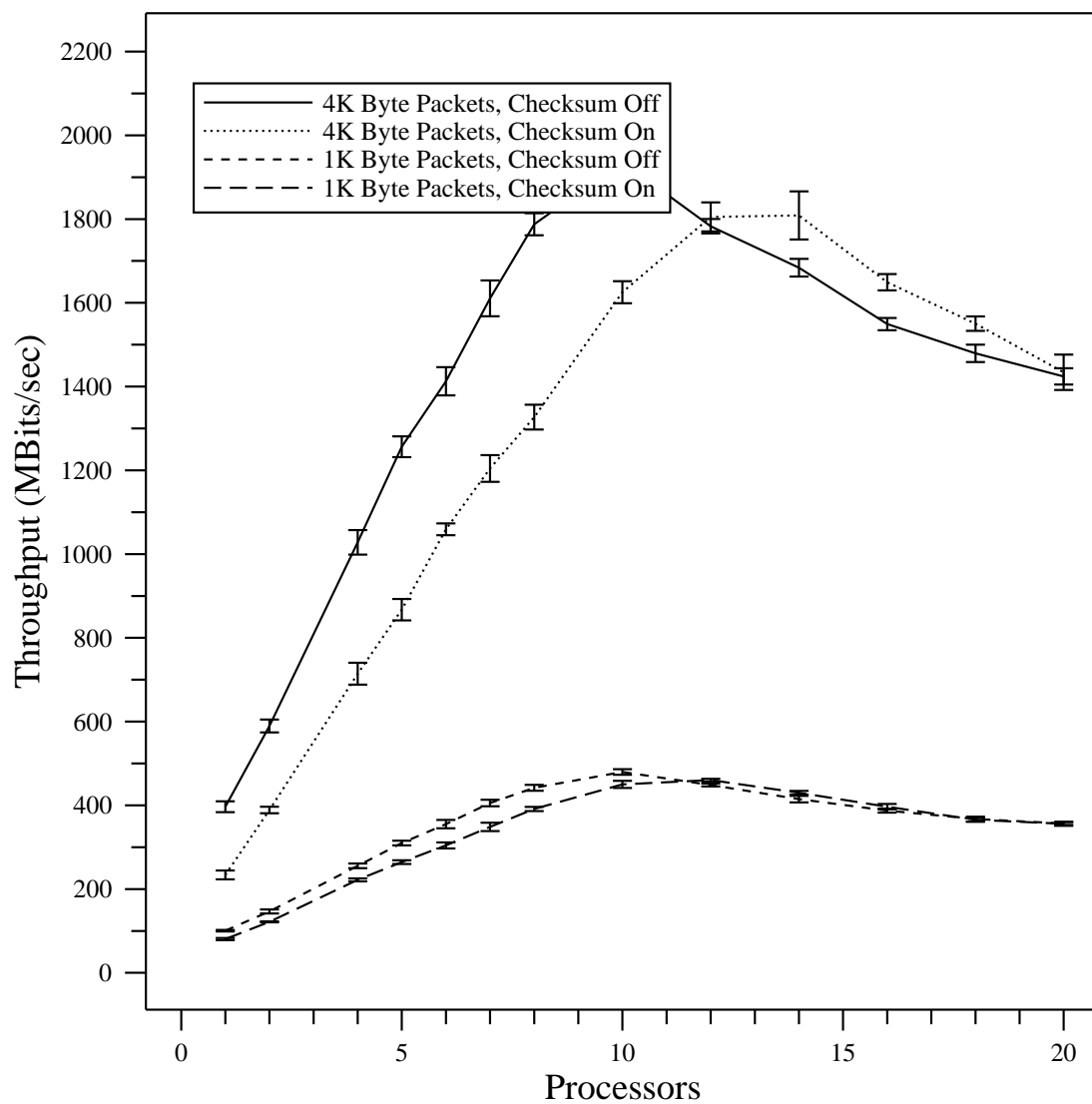
## Multiple Connections

Send & Recv, 4KB Packets, Cksum On & Off



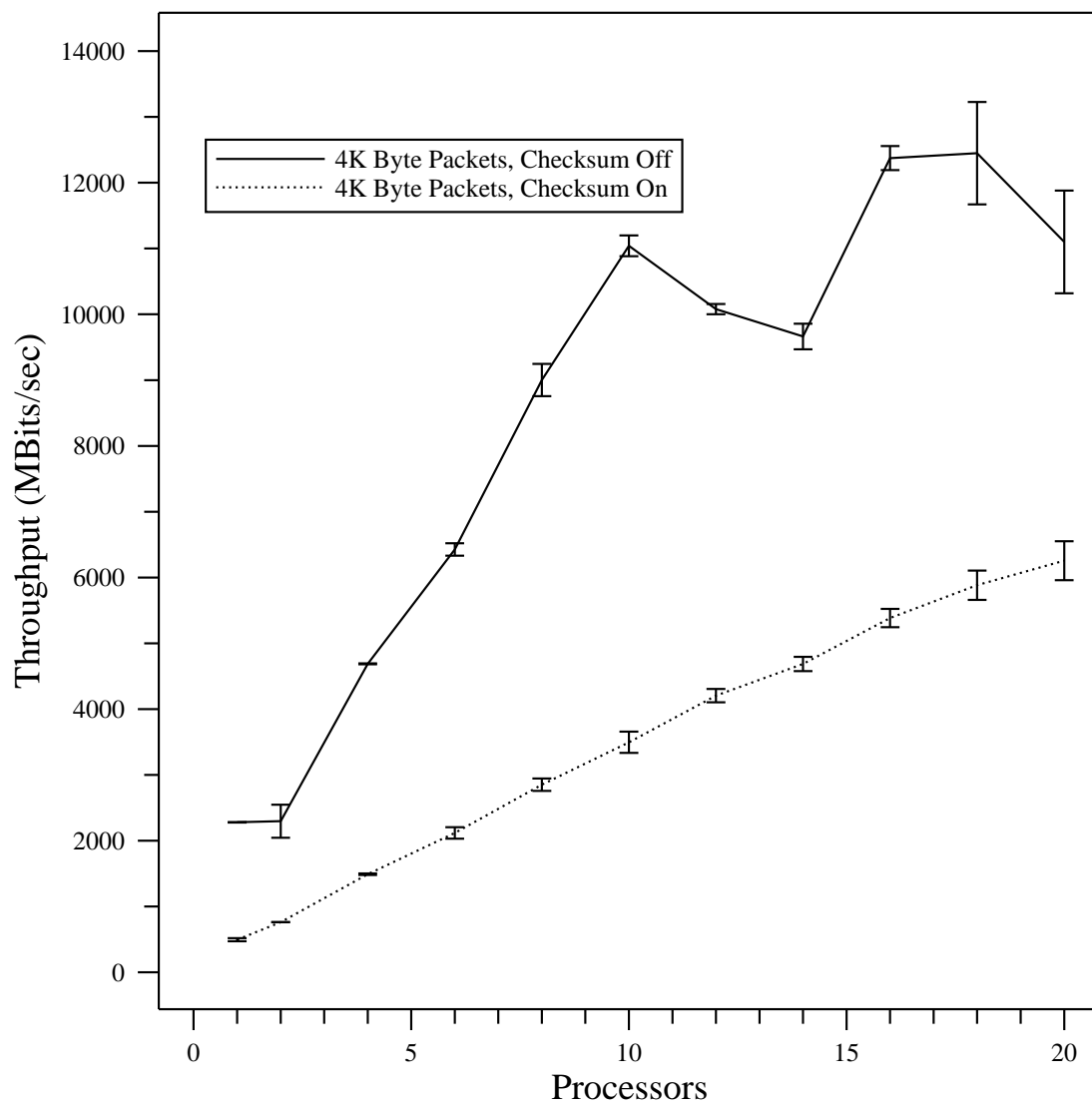
## UDP Send Side

Send Side, 4KB Packets, Cksum On & Off



## UDP Send Side (Take 2)

Send Side, 4KB Packets, Cksum On & Off



# UDP Receive Side

Recv Side, 4KB Packets, Cksum On & Off

