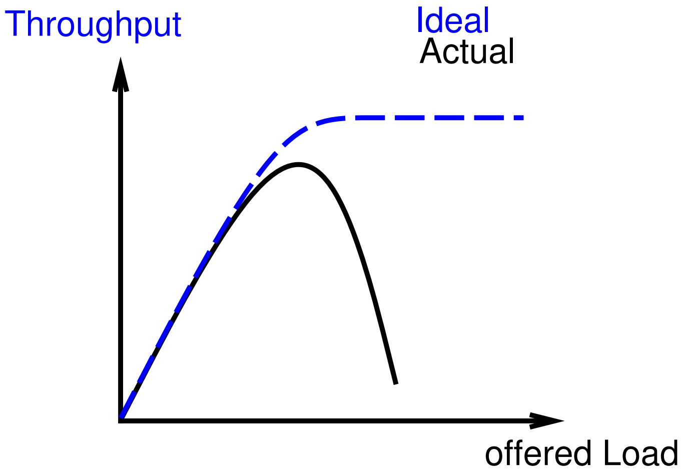
Lazy Receiver Processing (LRP): A Network Subsystem Architecture for Server Systems

Peter Druschel and Gaurav Banga

Department of Computer Science
Rice University

UNIX based Network Subsystems

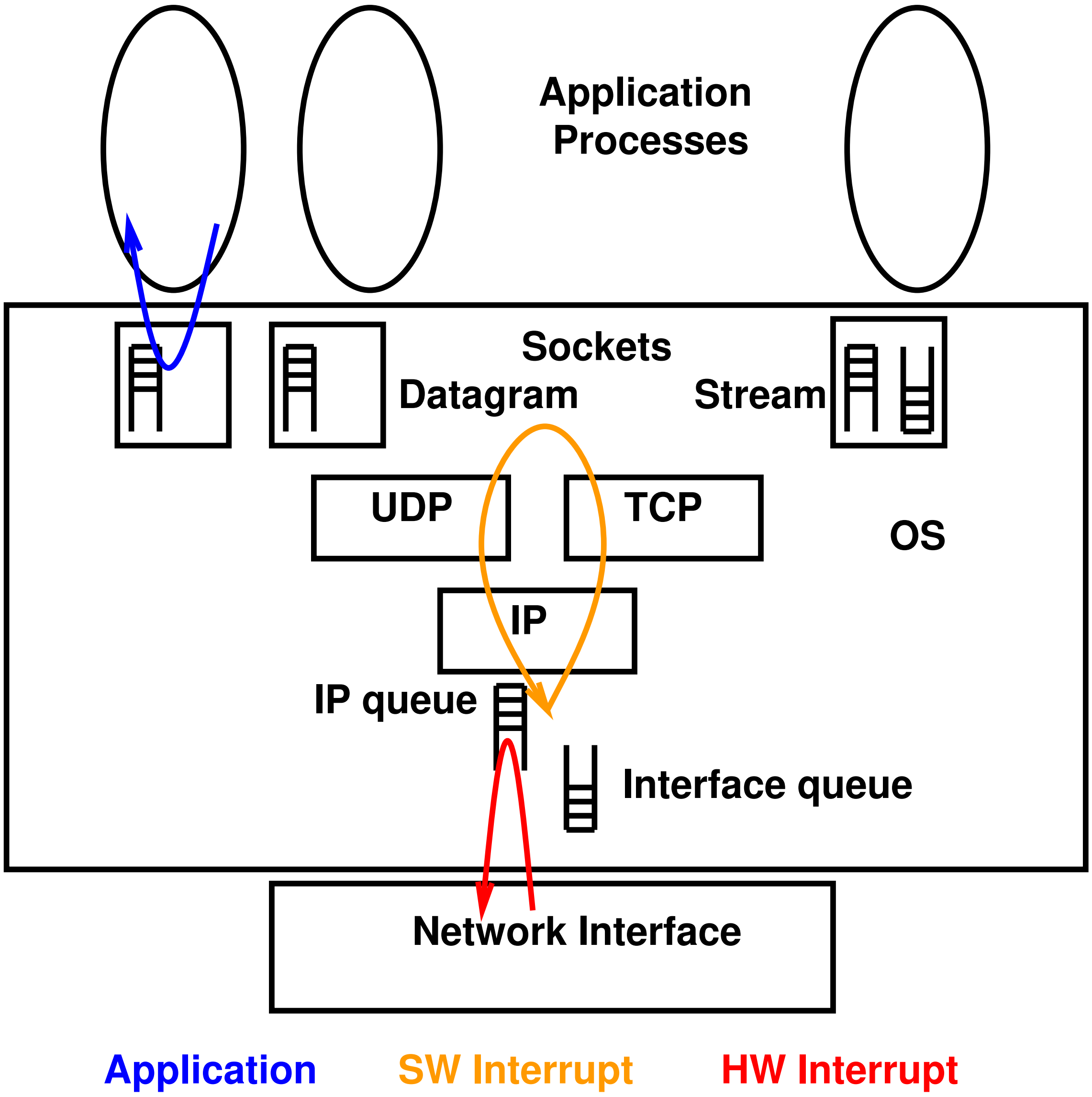


- stability
- fairness

Resource Management

- resources consumed by (on behalf of) application processes
- fairness and stable overload behavior
  - dynamic CPU scheduling policy
  - virtual memory page replacement policy
  - swapping policy
- **Problem:** incoming network traffic
  - resource use uncontrolled/unaccounted
  - cannot be immediately attributed to any application process

BSD UNIX Network Subsystem



Application Processes

Sockets

Datagram

Stream

UDP

TCP

OS

IP

IP queue

Interface queue

Network Interface

Application

SW Interrupt

HW Interrupt

BSD UNIX Network Subsystem

Problems

- **Eager receiver processing:** packet processing always preempts currently executing application
- **Lack of effective load shedding:** packet discard after significant host resources have been invested
- **Lack of traffic separation:** Application A's traffic can lead to delay and loss Application B's traffic
- **Inappropriate resource accounting:** Protocol processing charged to *interrupted* application

Sources of Network Overload

Is overload a problem in practice?

- misbehaved distributed applications
- simultaneous requests from large clients population (WWW)
- malicious denial-of-service attacks
- broadcast/multicast traffic
- incorrect client protocol implementations

Network capacity

- FDDI: almost 100K pkts/sec (actual)
- 155 Mbps ATM: $>$ 300K pkts/sec (theoretical)

Importance of...

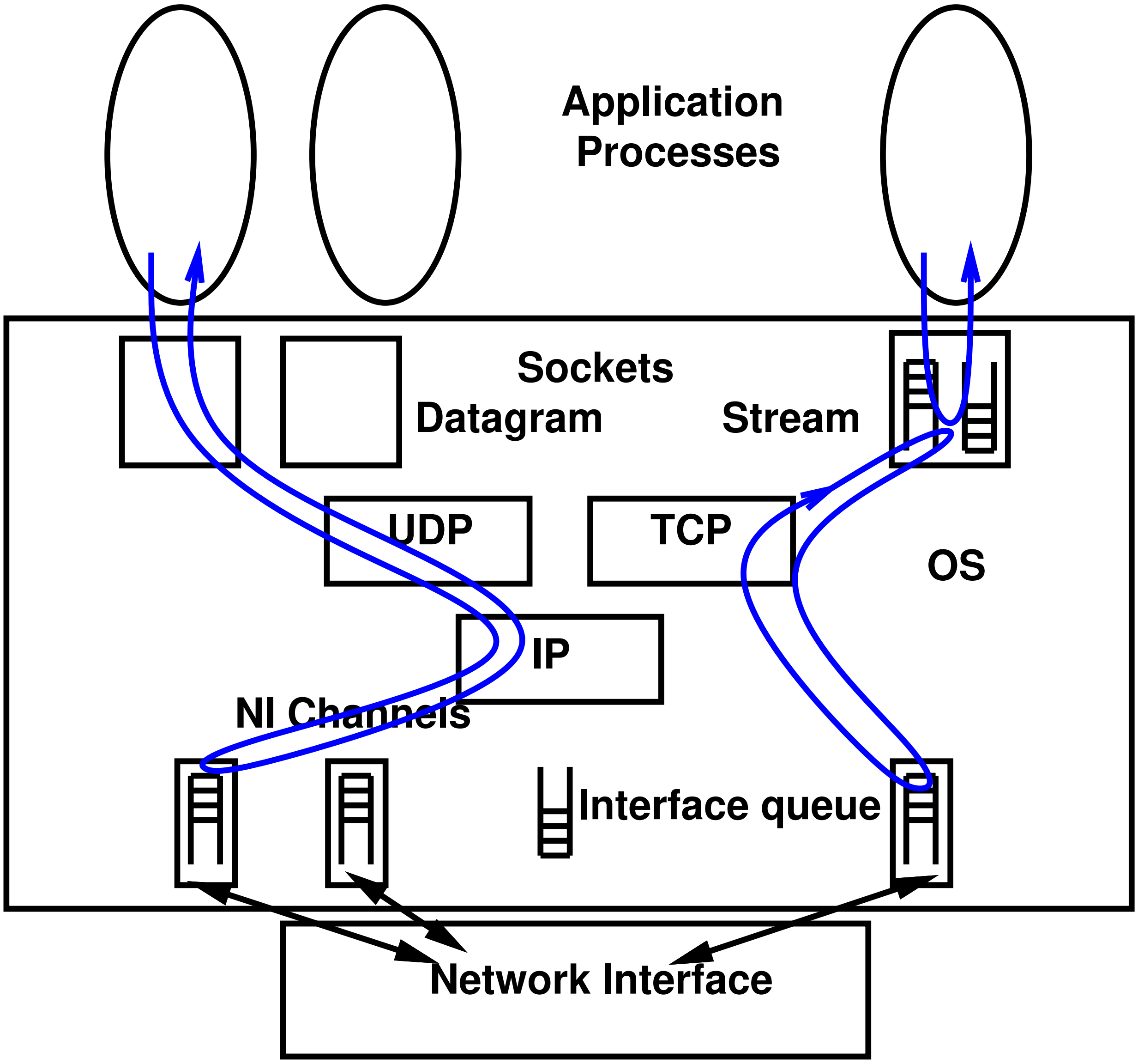
Overload stability

- any networked machine

Fairness

- multi-process/multi-threaded network servers
- application-level gateways
- soft real-time communication (multimedia)

Lazy Receiver Processing (LRP)



Lazy Receiver Processing (LRP)

- Per-socket receive queue (NI channel)
- NI demuxes/places incoming network packets (early demux)
- NI discards overflow packets (early discard)

Lazy Receiver Processing (LRP)

Protocol Processing performed...

- at priority of receiver
- lazily (just in time)
- in context of receiver (when possible)
- asynchronously (otherwise)

Lazy Receiver Processing (LRP)

- Packet arrival preempts executing process iff receiver has higher priority
  - no priority inversion
  - reduced context switching
  - increased locality
- Early demux plus processing at receiver priority
  - early packet discard
  - stable overload behavior
  - traffic separation
- Protocol processing in context of receiver
  - resources charged to receiver
  - fair resource allocation

Lazy Receiver Processing (LRP)

Mutually dependent techniques

- early demux/early discard
- lazy processing

1. Lazy processing depends on early demux
2. Early demux/discard alone insufficient
    - incomplete overload protection
    - interference
    - priority inversion

LRP and UDP

- Synchronous protocol processing
- In context of “receive” system call

Does LRP increase delay of UDP packets?

- Only if a CPU is idle at some time between arrival and “receive” invocation.
- Solution: do protocol processing “in idle loop”

LRP and TCP

- Need timely ACK processing to pace transmission
- Overlap TCP transmission / application processing

Solution: **Asynchronous TCP protocol processing**

- at priority of receiver
- “soon” (as soon as scheduled)
- if required per TCP state

LRP and TCP

Implementation

- similar to signal delivery
- at priority/on account of receiver
- run on per-CPU stack
- minimal per-socket overhead

Packet Demultiplexing

Two approaches:

- In network interface (NI Demux)
  - perfect overload behavior
  - requires NI demux support or embedded CPU
- In host interrupt handler (Soft Demux)
  - small per-arrival overhead on host
  - good overload behavior
  - works with any network interface

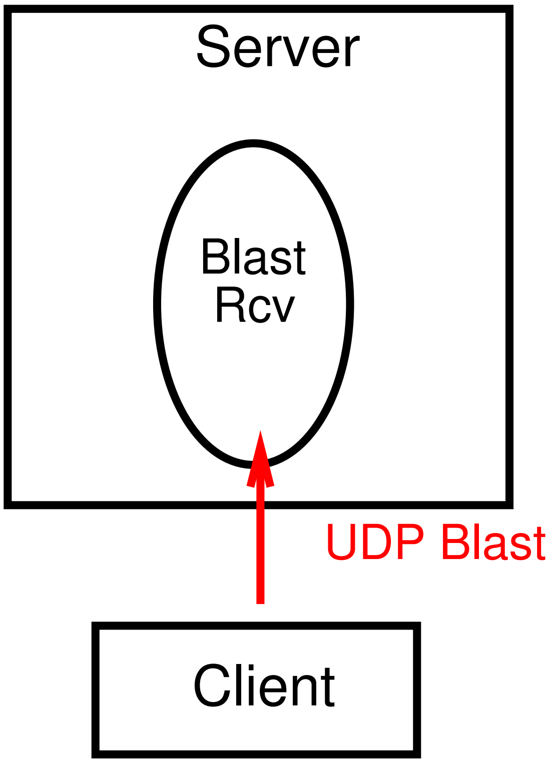
LRP Performance

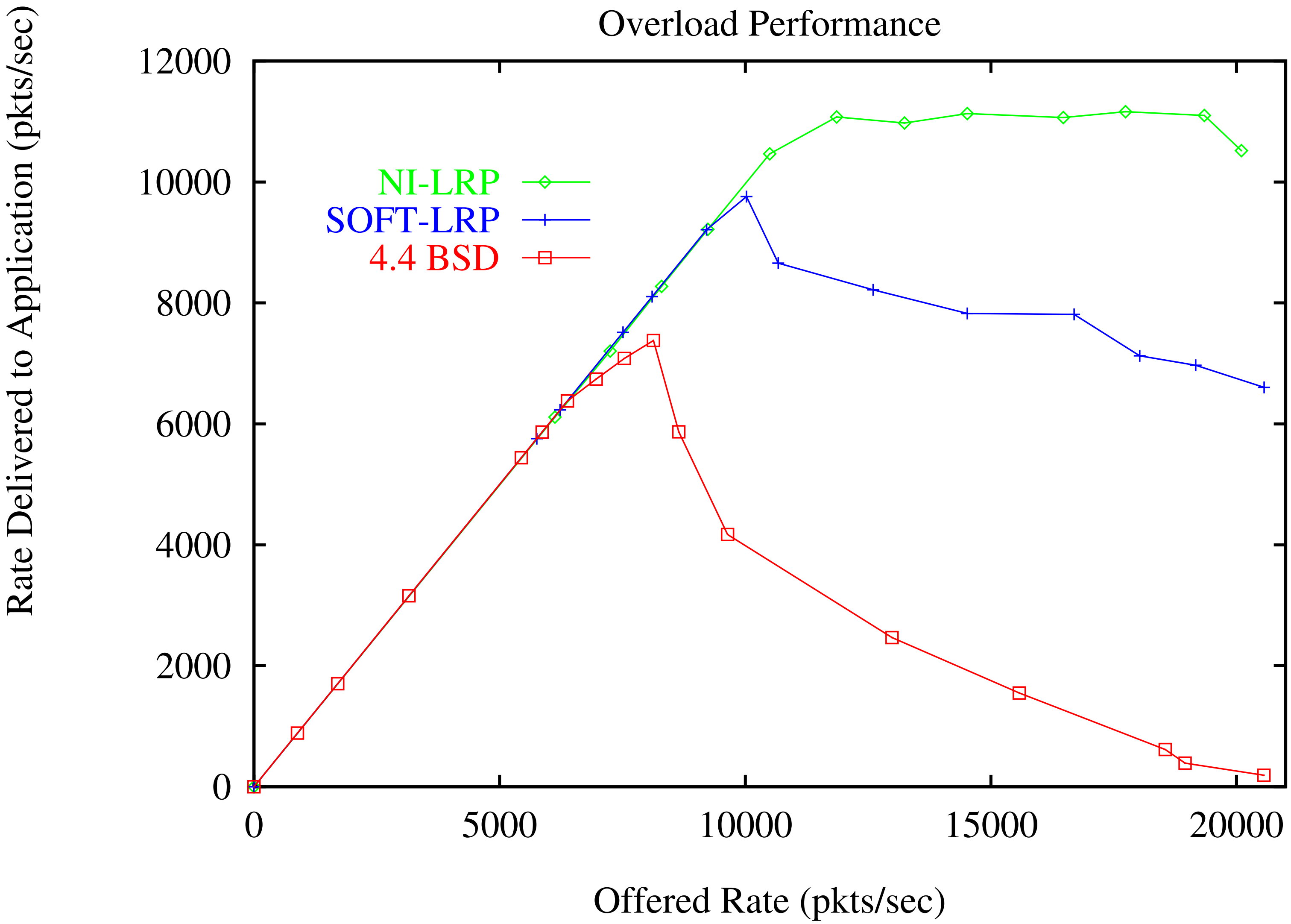
Experimental setup

- Sun SS20/61, 32Mbytes, SunOS 4.1.3_U1
- Fore SBA200 ATM network adaptors (155 Mbps)
- LRP network subsystem based on modified BSD4.4-Lite code

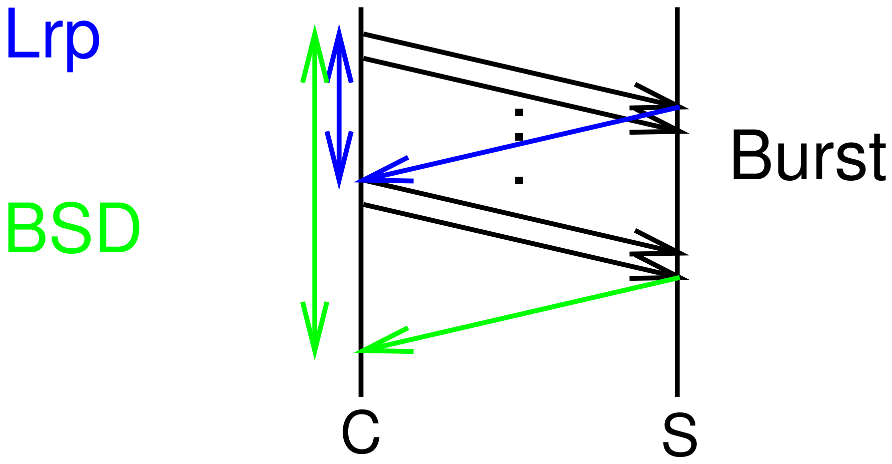
System	RT delay (μsecs)	UDP xput (Mbps)	TCP xput (Mbps)
SunOS, Fore driver	1006	64	63
4.4 BSD	855	82	69
LRP (NI Demux)	840	92	67
LRP (Soft Demux)	864	86	66

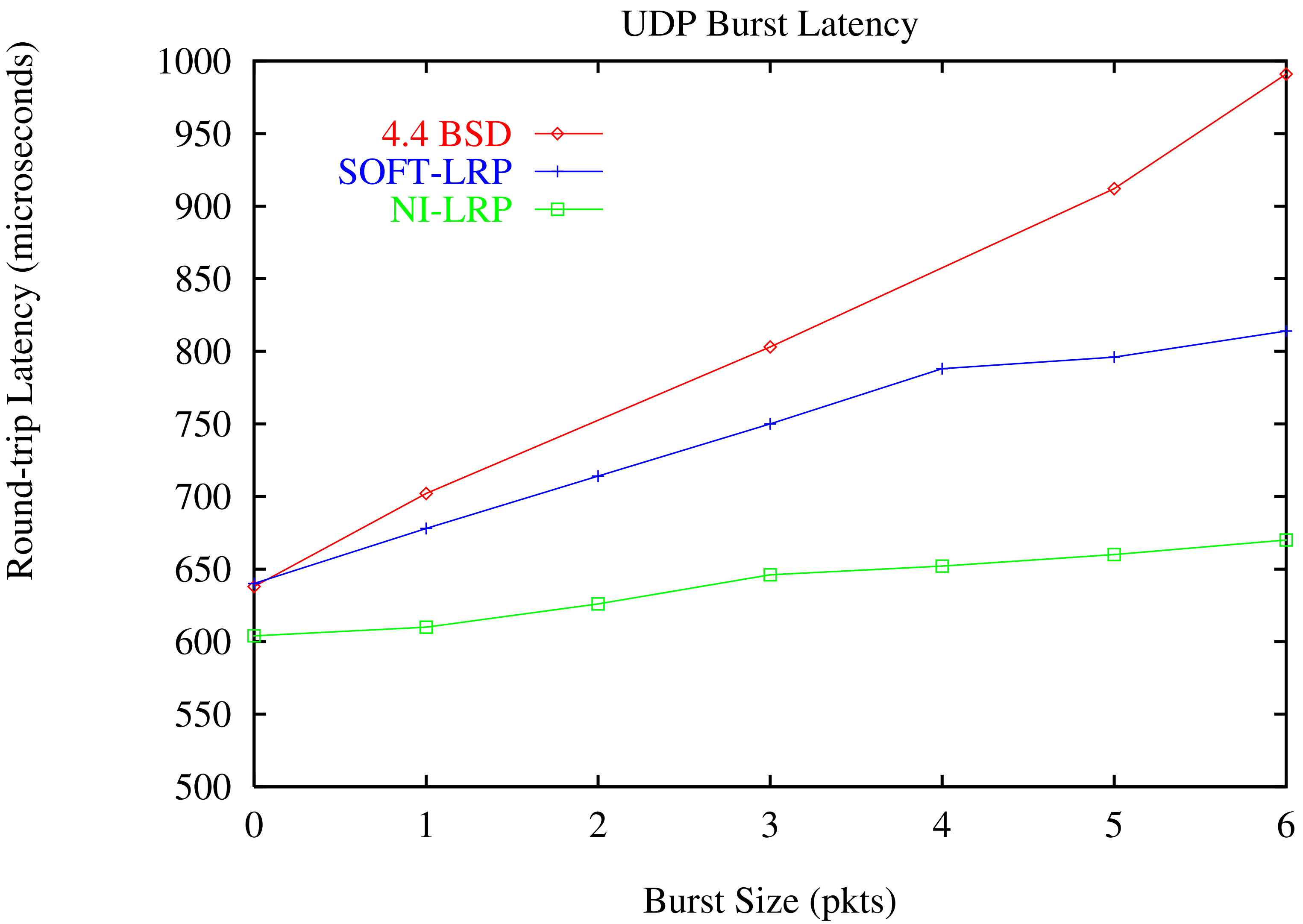
Overload



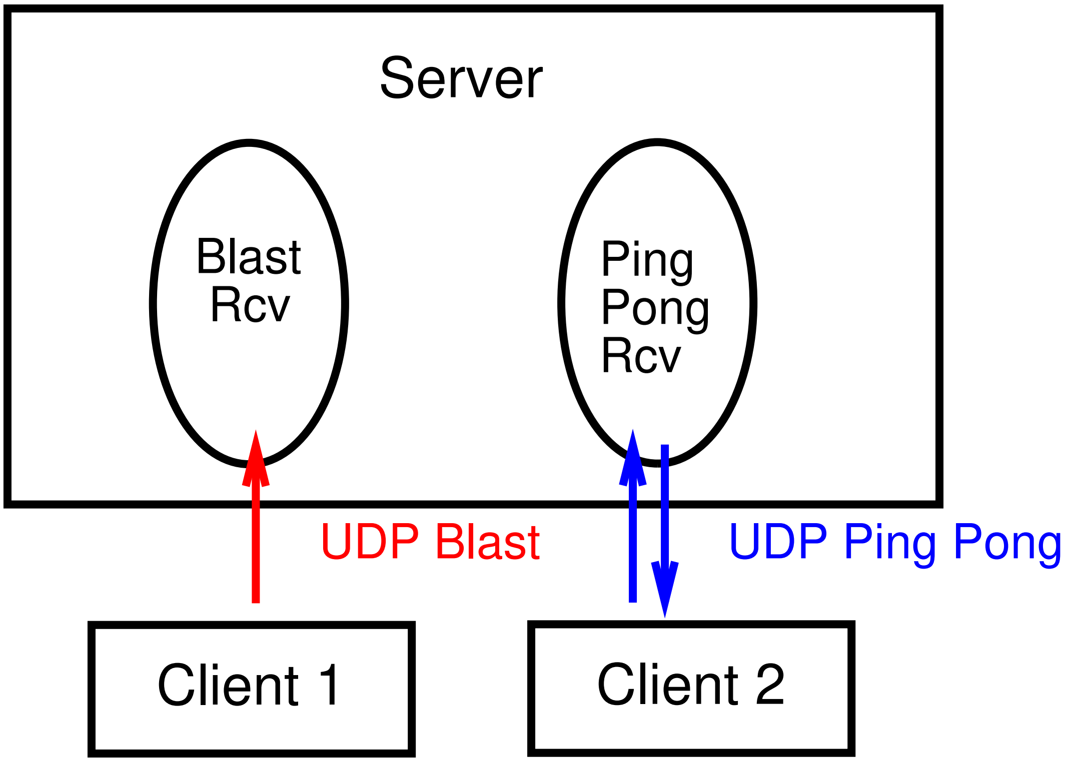

Overload Performance
Rate Delivered to Application (pkts/sec)
Offered Rate (pkts/sec)
NI-LRP
SOFT-LRP
4.4 BSD
0
2000
4000
6000
8000
10000
12000
0
5000
10000
15000
20000

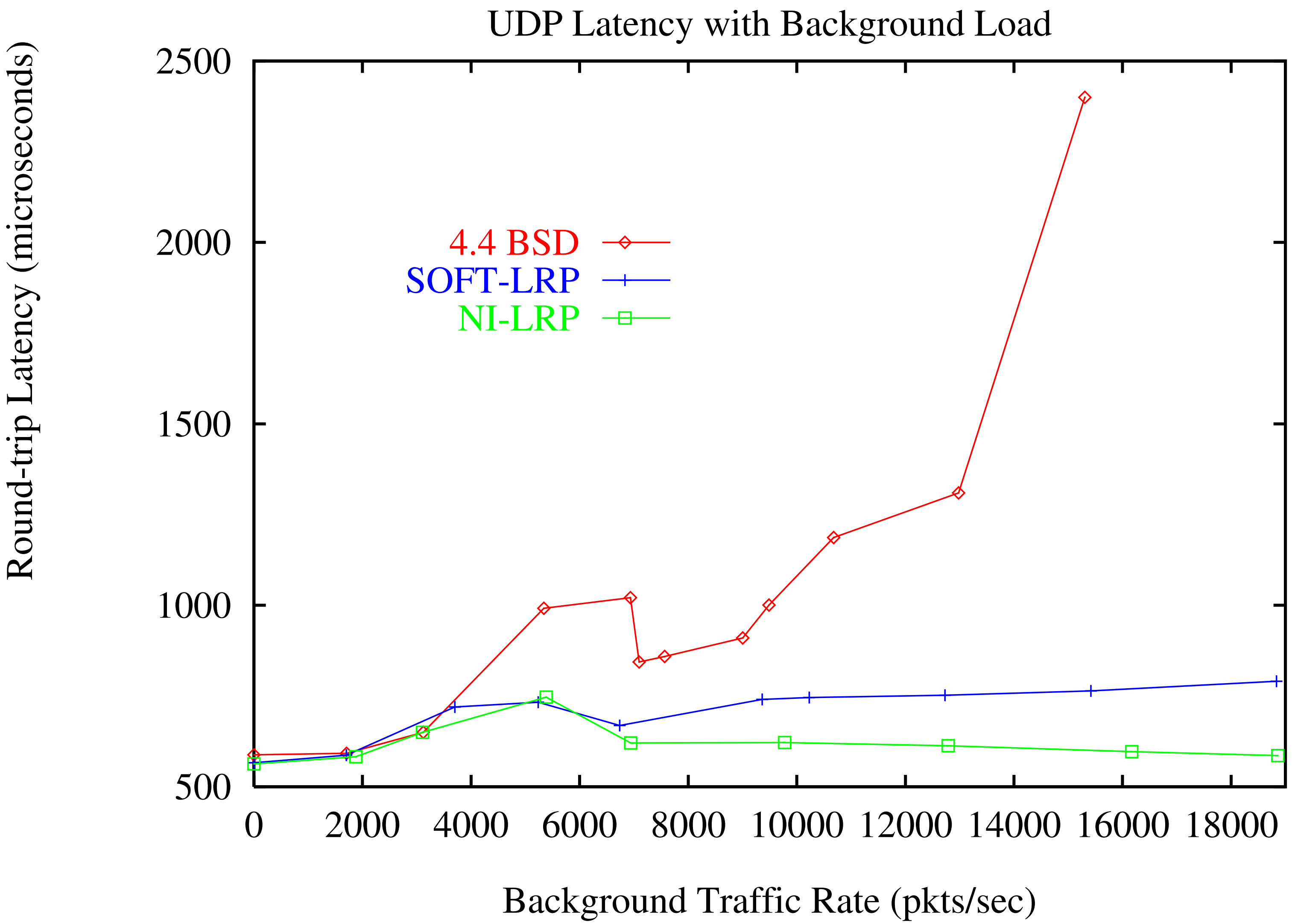
Burst Latency



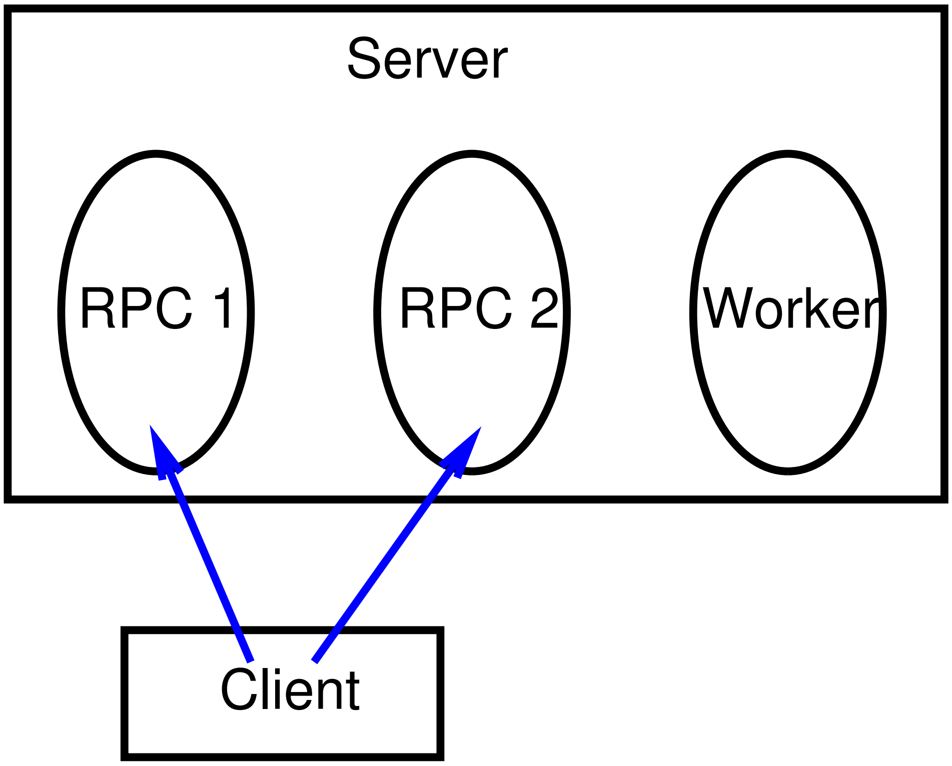

UDP Burst Latency
Round-trip Latency (microseconds)
1000
950
900
850
800
750
700
650
600
550
500
4.4 BSD
SOFT-LRP
NI-LRP
0
1
2
3
4
5
6
Burst Size (pkts)

Traffic Separation




UDP Latency with Background Load
Round-trip Latency (microseconds)
2500
2000
1500
1000
500
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
Background Traffic Rate (pkts/sec)
4.4 BSD
SOFT-LRP
NI-LRP

Synthetic Server Benchmark

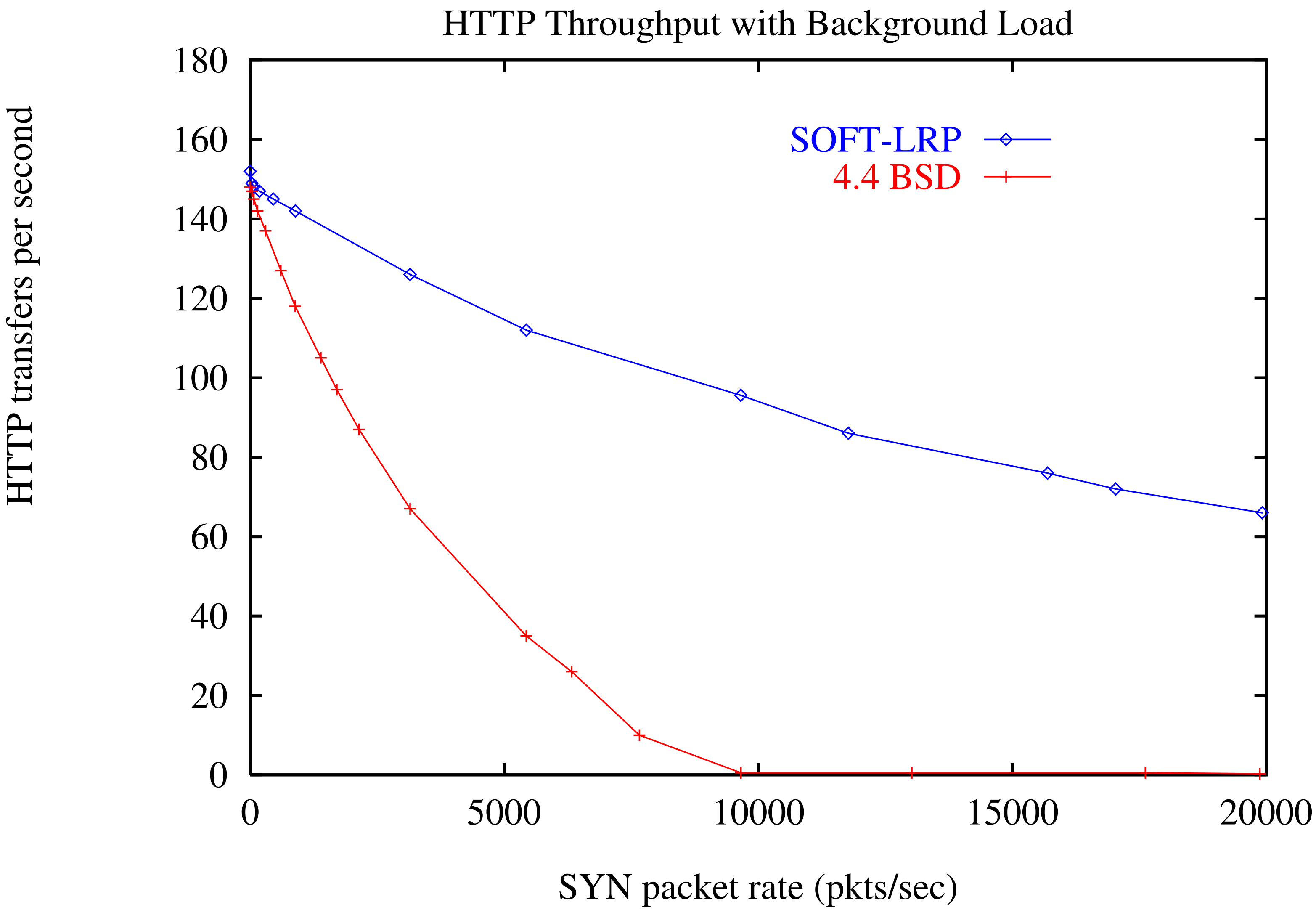


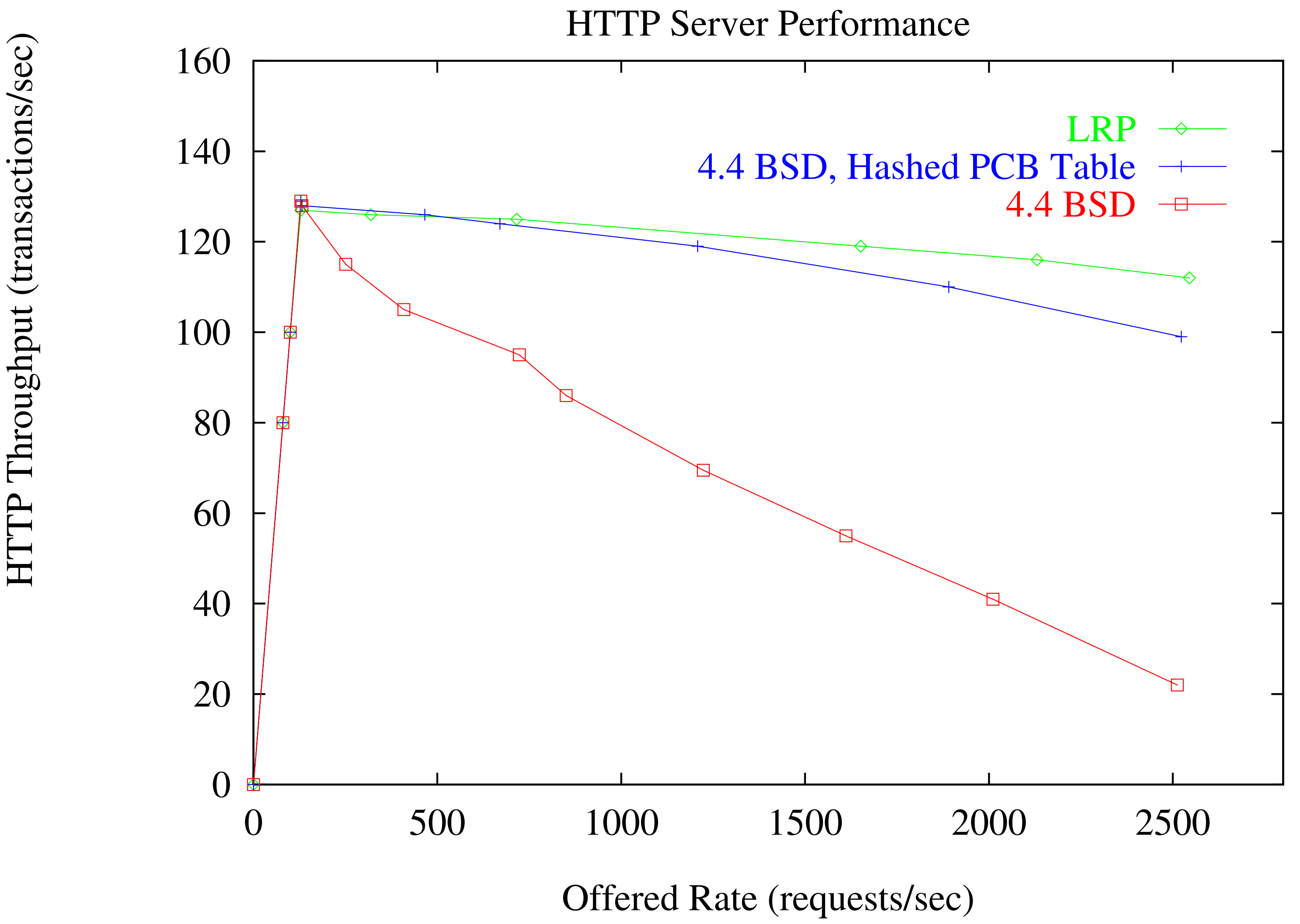
- Worker completion time:
  12–30% lower than BSD
- RPC rate: 0-10% higher than BSD
- Worker CPU share:
  29-33% vs. 23–26% with BSD

Synthetic Server Benchmark

RPC	System	Worker elapsed time (secs)	Server (RPCs/sec)
Fast	4.4BSD	49.7	3120
	SO-LRP	38.7	3133
	NI-LRP	34.6	3410
Medium	4.4BSD	47.1	2712
	SO-LRP	37.9	2759
	NI-LRP	34.1	2783
Slow	4.4BSD	43.9	2045
	SO-LRP	38.5	2134
	NI-LRP	35.7	2208

Table 1: Synthetic RPC Server Workload


HTTP Throughput with Background Load
HTTP transfers per second
180
160
140
120
100
80
60
40
20
0
SOFT-LRP
4.4 BSD
0
5000
10000
15000
20000
SYN packet rate (pkts/sec)


HTTP Server Performance
HTTP Throughput (transactions/sec)
Offered Rate (requests/sec)
0
20
40
60
80
100
120
140
160
0
500
1000
1500
2000
2500
LRP
4.4 BSD, Hashed PCB Table
4.4 BSD

Conclusion

LRP provides

- Stable overload behavior
- Increased throughput under high load
- Traffic separation
  - time
  - space

Efficient early demux/discard necessary for fairness under overload

Prototype implementation based on SunOS with BSD4.4 networking code: "http://www.cs.rice.edu/CS/Systems/LRP/"